Modeling the Supercritical Carbon Dioxide Brayton Cycle with Recompression

By

John J. Dyreby

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Mechanical Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

2014

Date of final oral examination: 8/1/2014

The dissertation is approved by the following members of the Final Oral Committee:

Gregory F. Nellis, Professor, Mechanical Engineering
Sanford A. Klein, Professor, Mechanical Engineering
Douglas T. Reindl, Professor, Mechanical Engineering
Mark H. Anderson, Research Professor, Engineering Physics
Michael L. Corradini, Professor, Engineering Physics

To Alissa – thanks for your patience.

Abstract

Supercritical carbon dioxide (SCO_2) power cycles show promise for a wide range of applications, such as concentrating solar power, next-generation nuclear reactors, and waste-heat recovery. Models capable of predicting the design-point, off-design, and part-load performance of SCO_2 power cycles are necessary for evaluating cycle designs. These models should be flexible in order to accommodate the range of designs under consideration and computationally efficient in order to enable timely optimization studies, possibly while considering cycle performance on an annual or life-cycle basis. This document reports on the development of a modeling framework that accommodates these requirements and is capable of predicting the performance of recuperated and recompression cycle configurations. The modeling framework is in Fortran and is flexible with respect to component-level specifics, such as the type of compressor used in the cycle or the method used to represent the off-design performance of the turbine. Optimization routines are integrated into the models, allowing exploration of optimal component and system designs or optimal operating strategies for a given system design.

The optimal design-point and off-design performance of various cycle designs is predicted using turbomachinery models based on the radial compressors and turbines that are currently being investigated by Sandia National Laboratory for use in SCO_2 applications. A range of heat rejection (low-side) temperatures are considered and results indicate that operating the cycles at warmer low-side temperatures requires a corresponding increase in low-side pressure in order to maximize thermal efficiency. The relationship between low-side temperature and pressure suggests that inventory control (i.e., actively controlling the low-side pressure) is a favorable control mechanism, especially if the power plant is expected to operate away from its design point for significant periods of time. For cycles designed to operate at warmer heat rejection temperatures (e.g., a dry-cooled design in an arid climate), the benefits of recompression are reduced and a simple recuperated cycle may be favorable. The optimal SCO_2 Brayton cycle design depends on the application being considered, and the developed modeling framework provides the consistent performance predictions that are required for further application-specific analyses.

Table of Contents

Abstract ii
Table of Contents iii
List of Figures iv
List of Tables vii
Nomenclature viii
1. Introduction 1
2. Design-Point Modeling Methodology 6
2.1. Turbomachinery 7
2.2. Heat Exchangers 8
2.3. Cycle Model 10
2.4. Fluid Properties 13
3. Off-Design Modeling Methodology 14
3.1. Component Model Interfaces 14
3.2. Sandia National Laboratory SCO_2 Compressor 17
3.3. Low-Reaction Radial Turbine 29
3.4. Heat Exchangers 34
3.5. Cycle Model 36
4. Modeling Framework 43
4.1. `core` Module 44
4.2. `design_point` Module 45
4.3. `off_design_point` Module 47
4.4. `heat_exchangers` Module 51
4.5. `compressors` Module 51
4.6. `turbines` Module 52
4.7. `CO2_properties` Module 52
5. Design-Point Analysis 53
5.1. Comparison to Literature 53
5.2. Effect of High-Side Pressure Limit 57
5.3. Effect of Low-Side Temperature 61
5.4. Convergence Verification 67
6. Off-Design Analysis 70
6.1. Varying Compressor Inlet Temperature 78
6.2. Varying Turbine Inlet Temperature 86
6.3. Cycle Performance with Alternative Turbine Model 88
6.4. Convergence Verification 94
7. Conclusions 96
7.1. Design-Point Considerations 96
7.2. Off-Design Considerations 97
7.3. Recommendations for Further Study 98
References 100
Appendix I: `core` Module Source Code 103
Appendix II: `design_point` Module Source Code 108
Appendix III: `off_design_point` Module Source Code 120
Appendix IV: `compressors` Module Source Code 134
Appendix V: `turbines` Module Source Code 144
Appendix VI: `heat_exchangers` Module Source Code 149
Appendix VII: `CO2_properties` Module Source Code 150
Appendix VIII: Recorded Data Plots for SNL Compression Test Cases 152

List of Figures

Figure 1.1: Diagram of a simple Brayton cycle (a) and a recompression cycle (b). Sandia National Laboratory has demonstrated a flexible 250 kW test loop that is capable of running in either configuration (Conboy et al., 2012). 2

Figure 2.1: Diagram of a recompression Brayton cycle, with design-point cycle model inputs shown in bold. 6

Figure 2.2: Iteration process for the design-point cycle model. 11

Figure 3.1: Performance map for the main compressor in the Sandia National Laboratory supercritical CO_2 test loop, taken from Wright et al., (2010). Note: the y-axis label should be "Corrected Specific Ideal Enthalpy Rise (BTU/lbm)"....18

Figure 3.2: Non-dimensional head-flow and efficiency-flow curves for the main compressor in the Sandia National Laboratory supercritical CO_2 test loop; points are colored by shaft speed. 20

Figure 3.3: Modified non-dimensional head-flow and efficiency-flow curves for the main compressor in the SNL supercritical CO_2 test loop. 22

Figure 3.4: Polynomial fits of the modified non-dimensional head-flow and efficiency-flow curves for the main compressor in the SNL supercritical CO_2 test loop. 22

Figure 3.5: Deviations in efficiency and ideal enthalpy rise associated with using the modified dimensionless numbers and polynomial fits. 23

Figure 3.6: Recorded measurement data for the Map Reference test case. 26

Figure 3.7: Recorded mass flow rate and pressure rise through the compressor for the Map Reference test case where the regions highlighted in red are assumed as steady-state operation. 27

Figure 3.8: Measured and predicted modified ideal head coefficient for the four test cases. 28

Figure 3.9: Efficiency of an ideal radial turbine as a function of velocity ratio. 30

Figure 3.10: Performance map for a turbine used in the SCO_2 test loop at Sandia National Laboratories, taken from the presentation associated with Wright et al. (2011). 31

Figure 3.11: Calculated effective nozzle area using data from the turbine performance map; the markers are colored by shaft speed. 32

Figure 3.12: Calculated effective nozzle area using data from the turbine performance map and inlet fluid density; the markers are colored by shaft speed. 33

Figure 3.13: Efficiency predicted from the performance map as a function of the ratio of tip speed to spouting velocity. 33

Figure 3.14: Head-flow and flow resistance curves for different shaft speeds and turbine inlet temperature; the curves intersect at the corresponding operating point of the cycle. 37

Figure 3.15: Diagram of a single-shaft recompression Brayton cycle (a) and a split-shaft recompression Brayton cycle (b); model inputs are shown in bold. 38

Figure 3.16: Iteration process for the off-design cycle model. 41

Figure 4.1: Iteration strategy for maximizing off-design cycle efficiency given a target power output or rate of heat addition. 50

Figure 5.1: Thermal efficiency as a function of recompression fraction for the "Basic" design with fixed low-side and high-side pressures of 7.69 MPa and 20 MPa, respectively. 54

Figure 5.2: Thermal efficiency as a function of recompression fraction for the "High Performance" design with fixed low-side and high-side pressures of 7.69 MPa and 20 MPa, respectively. 55

Figure 5.3: Thermal efficiency as a function of recompression fraction for the "Dry Cooled" design with fixed low-side and high-side pressures of 7.69 MPa and 20 MPa, respectively. 56

Figure 5.4: Optimal thermal efficiency for a SCO_2 cycle with a compressor inlet temperature of 32°C. The dashed line corresponds to a recompression fraction of zero (i.e., a simple cycle). The open circles mark the normalized conductance corresponding to a minimum temperature difference of 10°C in the recuperators while the filled circles correspond to a 2°C minimum temperature difference. 58

Figure 5.5: Optimal thermal efficiency for a SCO_2 cycle with a compressor inlet temperature of 55°C. The dashed line corresponds to a recompression fraction of zero (i.e., a simple cycle). The open circles mark the normalized conductance corresponding to a minimum temperature difference of 10°C in the recuperators while the filled circles correspond to a 2°C minimum temperature difference. 59

Figure 5.6: Optimal compressor inlet and outlet pressures for the 32°C case (left) and the 55°C case (right). The polytropic efficiency of the turbomachinery is 0.9. 60

Figure 5.7: Thermal efficiency as a function of compressor inlet temperature for various designs. 62

Figure 5.8: Optimal low-side pressure (top), recompression fraction (middle), and low-temperature recuperator conductance fraction (bottom) as a function of low-side temperature with a high-side temperature of 700°C. 63

Figure 5.9: Thermal efficiency as a function of compressor inlet temperature for various designs assuming a one percent relative pressure drop through the heat exchangers in the cycle. 65

Figure 5.10: Thermal efficiency as a function of compressor inlet temperature for various designs assuming a two percent relative pressure drop through the heat exchangers in the cycle. 65

Figure 5.11: Optimal low-side pressure (top), recompression fraction (middle), and low-temperature recuperator conductance fraction (bottom) as a function of low-side temperature with a high-side temperature of 700°C and a one percent relative pressure drop through the heat exchangers in the cycle. 66

Figure 5.12: Relative error of thermal efficiency (black lines) and mass flow rate (blue lines) for a number of design points. The reference values used for the calculating errors are determined from runs using 30 sub-heat exchangers and a convergence tolerance of 10^{-8}. 68

Figure 5.13: Design-point cycle model runtime using FIT (bottom) and REFPROP (top) for two representative designs. 69

Figure 6.1: Power output and corresponding compressor outlet pressure as a function of compressor inlet pressure for various off-design temperatures. The black dots represent the design points for the two cycles, and the dashed line indicates surge in the main compressor. Note that these results do not take into account physical limitations of the turbomachinery. 71

Figure 6.2: Ratio of turbomachinery tip speed to local speed of sound for various off-design low-side temperatures and pressures; values greater than one are not physically valid. 73

Figure 6.3: Power output and compressor outlet pressure for various off-design conditions. Only valid operating points are plotted. 74

Figure 6.4: Ratio of turbomachinery tip speed to local speed of sound for various off-design low-side temperatures and pressures. 75

Figure 6.5: Predicted power output, high-side pressure, and thermal efficiency for the low-temperature and high-temperature recompression designs. The solid lines indicate operation at maximum efficiency, the dashed lines indicate operation at maximum power output, and the dotted lines are operation at maximum efficiency using design-point shaft speed. 79

Figure 6.6: Predicted power output, high-side pressure, and thermal efficiency for the low-temperature and high-temperature simple designs. The solid lines indicate operation at maximum efficiency, the dashed lines indicate operation at maximum power output, and the dotted lines are operation at design-point shaft speed. 80

Figure 6.7: Thermal efficiency of the four designs as a function of off-design compressor inlet temperature. 81

Figure 6.8: Control parameters associated with optimal performance under off-design compressor inlet temperature for the two recompression cycle designs. 83

Figure 6.9: Control parameters associated with optimal performance under off-design compressor inlet temperature for the two simple cycle designs. 84

Figure 6.10: Head coefficient for the compressor and recompressor and turbine spouting velocity ratio under off-design conditions for the recompression designs. 85

Figure 6.11: Head coefficient for the compressor and turbine spouting velocity ratio under off-design conditions for the simple designs. 85

Figure 6.12: Predicted power output, high-side pressure, and thermal efficiency for the low-temperature and high-temperature recompression designs. The solid lines indicate operation at maximum efficiency, the dashed lines indicate operation at maximum power output, and the dotted lines are operation at maximum efficiency using design-point shaft speed. 86

Figure 6.13: Predicted power output, high-side pressure, and thermal efficiency for the low-temperature and high-temperature simple designs. The solid lines indicate operation at maximum efficiency, the dashed lines indicate operation at maximum power output, and the dotted lines are operation at design-point shaft speed. 87

Figure 6.14: Thermal efficiency of the four designs as a function of off-design turbine inlet temperature. 88

Figure 6.15: Thermal efficiency of the three designs at the rated 10 MW power output as a function of off-design compressor inlet temperature. The design points are indicated with a circle and only achievable values (with a high-pressure limit of 30 MPa) are plotted. 91

Figure 6.16: Control parameters associated with the efficiencies predicted in Fig. 6.15. 91

Figure 6.17: Off-design thermal efficiency of the three shaft configurations for the 32°C and 50°C designs at the rated 10 MW power output. The design points are indicated with a circle and only achievable values (with a high-pressure limit of 30 MPa) are plotted. 93

Figure 6.18: Part-load thermal efficiency of the three shaft configurations for the 32°C and 50°C designs 93

Figure 6.19: Convergence of the `optimal_off_design` subroutine. 94

Figure 6.20: Model runtime for the four designs, using the `optimal_off_design` subroutine with compressor inlet pressure specified for a number of representative conditions. 95

List of Tables

Table 3.1: Required inputs and outputs for compatible off-design compressor models. 15

Table 3.2: Required inputs and outputs for compatible off-design recompressor models. 15

Table 3.3: Required inputs and outputs for compatible off-design turbine models. 15

Table 3.4: Required inputs and outputs for compatible off-design heat exchanger models. 16

Table 3.5: Inlet conditions for various carbon dioxide compression tests. 24

Table 4.1: Fortran modules used by the developed modeling framework. 44

Table 4.2: Description of input arguments used by the `design` subroutine. 45

Table 4.3: Description of input arguments used by the `optimal_design` subroutine. 46

Table 4.4: Input arguments to the off-design cycle model. 48

Table 4.5: Description of input arguments used by the `optimal_off_design` subroutine. 49

Table 5.1: Design-point conditions for model comparison with literature. 53

Table 6.1: Four designs of interest. 77

Table 6.2: Three designs of interest, characterized by their compressor inlet temperatures. 89

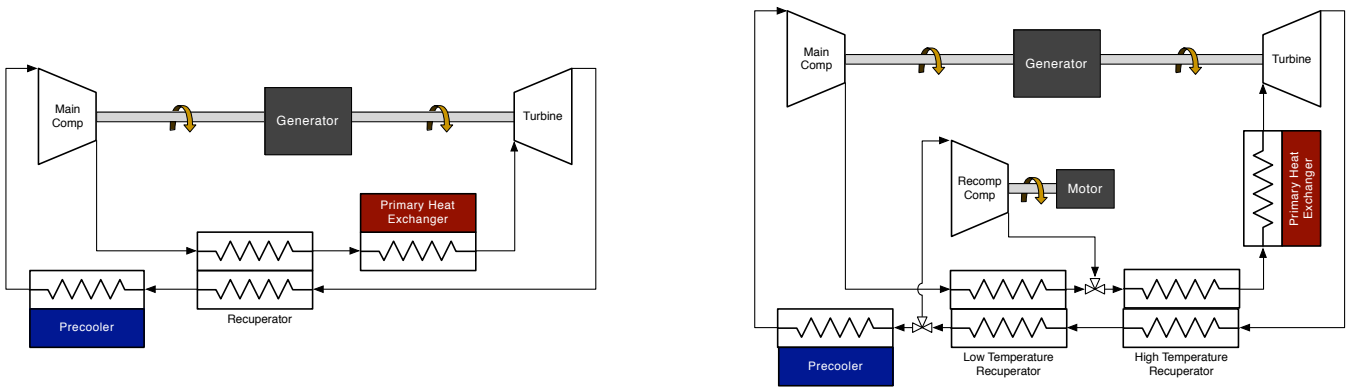
Nomenclature

ε	heat exchanger effectiveness
η	efficiency
η^*	modified efficiency
ν	ratio of turbine tip speed to spouting velocity
ρ	density
ϕ	compressor flow coefficient
ϕ^*	modified compressor flow coefficient
ϕ_{rc}	recompression fraction
ψ_i	compressor ideal head coefficient
ψ_i^*	modified compressor ideal head coefficient
Δh_i	isentropic specific enthalpy change
ΔP	pressure drop in heat exchanger
ΔP_{design}	design-point pressure drop in heat exchanger
f	friction factor
h_{in}	inlet specific enthalpy
$h_{out,i}$	outlet specific enthalpy if compression or expansion is isentropic
h_{out}	outlet enthalpy
$\dot{m}$	mass flow rate
s_{in}	inlet specific entropy
w	specific work
w_i	isentropic specific work
A_{nozzle}	turbine effective nozzle area
$\dot{C}$	capacitance rate
$\dot{C}_{\mathrm{min}}$	minimum capacitance rate
C_R	capacitance ratio between streams of heat exchanger
C_s	turbine spouting velocity
CSP	concentrating solar power
D	diameter
HT	high-temperature, referring to the high-temperature recuperator
LT	low-temperature, referring to the low-temperature recuperator
NTU	heat exchanger dimensionless number of transfer units
N	shaft speed
N_{design}	design-point shaft speed
N_{mc}	main compressor shaft speed
N_t	turbine shaft speed
$P_{mc,in}$	main compressor inlet pressure
RC	recompressing compressor, also referred to as the “recompressor”
SCO_2	supercritical carbon dioxide
SNL	Sandia National Laboratory
T	temperature
$T_{mc,in}$	main compressor inlet temperature
$T_{t,in}$	turbine inlet temperature
U	rotor tip speed in compressor or turbine
UA	heat exchanger conductance
UA_{design}	design-point heat exchanger conductance

1. Introduction

Brayton cycles operating with supercritical carbon dioxide (SCO_2) have the potential to offer improved thermal-to-mechanical conversion efficiency for utility scale electricity production. Supercritical carbon dioxide Brayton cycles are characterized by their operation above the critical temperature and pressure of carbon dioxide (31.1°C and 7.39 MPa, respectively) and were first proposed in 1967 by E. Feher (1967). A number of related publications followed, a thorough overview of which is presented by V. Dostal (2004), including an extensive summary by G. Angelino of the theoretical performance of various CO_2 cycle configurations (Angelino, 1968; Angelino, 1969). In his work, Angelino analyzed the design-point performance of a number of cycles designed to take advantage of the properties of carbon dioxide near its critical point. Two promising SCO_2 cycle configurations that emerged from his work are the simple Brayton cycle with recuperation, shown in Figure 1.1(a), and the recompression cycle, shown in Figure 1.1(b). The recompression cycle was chosen for further analysis by Dostal because it offers the potential for high efficiency without introducing a significant amount of complexity compared to the simple cycle. Despite showing promise, the cycles were never deployed due to the technical difficulties associated with the high pressures and temperatures that are required. From the mid 1970's to the late 1990's, there were no significant advancements or apparent interest in SCO_2 power cycles.

Since the late 1990's, advances in turbomachinery and heat exchanger design, as well as experience with higher operating temperatures and pressures in advanced steam-based Rankine power cycles, have resulted in a resurgence of interest in SCO_2 Brayton cycles. Dostal credits work done at the Czech Technical University in Prague, Czech Republic in 1997 as the catalyst for this revival of interest in SCO_2 power cycles. Since then, the unique advantages offered by SCO_2 Brayton cycles have been recognized by a number of industries, including nuclear and concentrating solar power (CSP), and publications related to SCO_2 research have risen dramatically in recent years.



(a) (b)

Figure 1.1. Diagram of a simple Brayton cycle (a) and a recompression cycle (b). Sandia National Laboratory has demonstrated a flexible 250 kW test loop that is capable of running in either configuration (Conboy et al., 2012).

The advantages of SCO_2 Brayton cycles make them well suited for a variety of applications. With heat source temperatures above 550°C, SCO_2 Brayton cycles have higher thermal efficiencies than superheated steam Rankine cycles (Ishiyama et al., 2008; Dostal, 2004). The low specific volume of carbon dioxide at the cycle's operating pressures translates into considerably smaller turbomachinery compared to steam-based Rankine cycles. Because there is no risk of carbon dioxide undergoing a phase change from gas to liquid as it expands through the turbine in a supercritical Brayton cycle, concerns over blade erosion due to droplet impingement are eliminated. Also, heat rejection for a SCO_2 Brayton cycle is not limited by the saturation temperature of the working fluid, offering the potential for cost-effective dry cooling (i.e., the use of ambient air as the sole heat rejection medium).

In recent years there has been a growing interest in dry cooling because water consumption due to thermoelectric power generation comprises a large portion of the overall water demands, and these demands on our water resources will continue to grow as population and energy use increase. Depending on the power plant type and cooling mechanism, between 200 and 900 gallons of water per MWhe are required to generate electricity (DOE, 2006). To put this into perspective, in 2005 the United States used 410 billion gallons of water, of which roughly 201 billion gallons (143 fresh, 58 saline) were withdrawn to generate electricity (Kenny et al., 2009). The amount of water that is required for heat rejection from

power cycles is especially disadvantageous to CSP applications, which are typically located in arid regions where water is scarce (Carter & Campbell, 2009). Because of the potential water and cost savings associated with dry cooling, a better understanding of the SCO_2 Brayton cycle's performance under dry-cooled conditions is warranted.

While a number of investigators have reported the theoretical thermodynamic, design-point performance of SCO_2 power cycles (Dostal et al., 2006; Ishiyama et al., 2008; Sarkar, 2009; Utamura, 2010; Chacartegui et al., 2011), the performance of heat exchangers in the cycle is typically represented using a heat exchanger effectiveness. A disadvantage of characterizing heat exchanger performance with an effectiveness is that identical values of effectiveness under different design-point conditions can result in heat exchangers of dramatically different sizes and costs. Representing the recuperators using conductance, rather than effectiveness, allows for a more appropriate comparison among design-point conditions because a larger conductance typically corresponds more directly to a physically larger and higher capital cost heat exchanger. Two notable exceptions in the literature to date are the overview of cycle performance provided by Bryant et al. (2011), which includes a figure relating thermal efficiency to recuperator conductance, and the analysis by Neises and Turchi (2013), which uses a conductance-based method in order to compare a number of cycle configurations at a single design point.

Despite the recent popularity of the SCO_2 Brayton cycle, few investigators have developed models that are capable of predicting off-design or part-load operation of the cycle. Such capabilities are required in order to analyze cycles on a seasonal or annual basis, especially in the context of dry cooling that will likely lead to the cycle operating away from its design point for extended periods of time. Off-design models have been developed by Argonne National Laboratory (ANL), which created a plant dynamics code to model an SCO_2 cycle coupled to a lead-cooled pool-type fast reactor (Moisseytsev & Sienicki, 2011), and Knolls Atomic Power Laboratory (KAPL), which developed a model using TRACE, the TRAC/RELAP Advanced Computational Engine provided by the U.S. Nuclear Regulatory Commission (Hexemer & Rahner, 2011). However, both of these models are specific to a particular application

(the ANL model couples the power block and reactor and the KAPL model is specific to the Integrated System Test demonstration loop) and both are designed to model transient performance with timescales on the order of seconds to minutes. Annual simulations using either of these models would be prohibitively time-consuming. The work by Dostal at MIT (Dostal, 2004) resulted in the CYCLES code, which evolved into CYCLES II (Legault, 2006) and CYCLES III (Ludington, 2009). The off-design steady-state capability of the CYCLES model is similar to the capability of the models developed under this work effort, but the work at MIT makes a number of assumptions about the balance-of-system associated with the power cycle that limits the cycle models applicability to other applications. For example, the shaft speed of the turbomachinery is fixed at 3,600 rpm and the cycle model is coupled with a model of its heat source, which is a sodium-cooled fast reactor (SFR). Also, the source code for CYCLES is not, to the author's best knowledge, available online for other investigators to use.

The goal of this work effort is to develop and make available versatile and computationally efficient models that allow the characterization and evaluation of recuperated and recompression SCO_2 Brayton power cycles under design-point and off-design conditions. The developed models are flexible with respect to component-level specifics, such as the type of compressor used in the cycle or the approach used to represent the off-design performance of the turbine. This flexibility is accomplished by providing well-documented interfaces to "black box" component models, allowing a user to represent components with any degree of complexity that is desired. Optimization routines are integrated into the models, allowing exploration of optimal component and system designs based on defined constraints (e.g. source and sink temperature) or optimal operating/control strategies for a given system design. Ultimately, the selection of the best-possible design for a given application will strongly depend on a number of economic variables. The selection and justification of these variables, which range from the capital costs of the components to the terms of the Power Purchase Agreement (PPA) used to govern the sale of the generated electricity, are beyond the scope of this work effort. Therefore, the core cycle models presented in this document are envisioned as building blocks for more complex and specific simulations. For example, the

models can easily be integrated into the TRNSYS simulation environment and combined with models developed for a specific power plant. Likewise, the cycle models can be coupled with models of various heat rejection mechanisms in order to investigate the effects of cooling-related parasitic losses on cycle design.

The organization of the remainder of this document is as follows. Chapters 2 and 3 describes the methodology developed to model SCO_2 systems under design-point and off-design conditions, respectively, including the off-design turbomachinery models that are based on SCO_2 compressors and turbines being investigated by Sandia National Laboratory (Conboy et al., 2012). The implementation of this methodology into Fortran code is explained in Chapter 4. A number of design-point considerations are investigated in Chapter 5, including the effects of high-side (compressor outlet) pressure and low-side (compressor inlet) temperature on cycle performance. Chapter 6 is an exploration of various off-design and part-load performance predictions, including a discussion on optimal control strategies for the cycles. General conclusions and recommendations for future applications of the modeling framework are considered in Chapter 7.

2. Design-Point Modeling Methodology

A diagram of the major components and their arrangement in the recompression cycle configuration is shown in Figure 2.1, with the design-point cycle model inputs shown in bold.

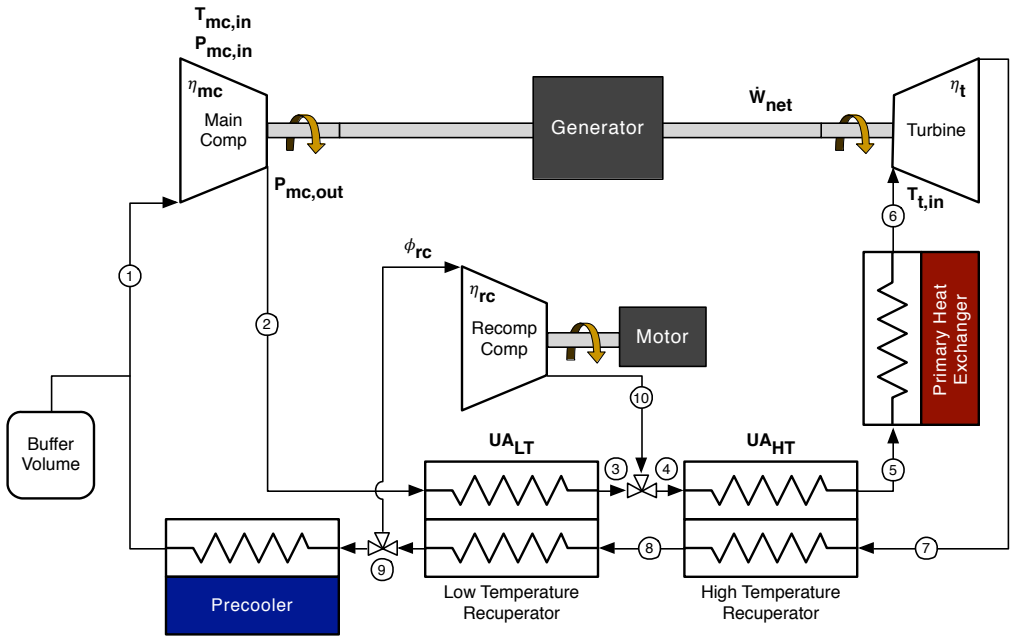


Figure 2.1. Diagram of a recompression Brayton cycle, with design-point cycle model inputs shown in bold.

The recompression cycle is similar to a closed Brayton cycle with recuperation[1], with the addition of a bypass that diverts a fraction of the CO_2 to a secondary compressor before the remainder of the stream enters the precooler. The diverted CO_2 is pressurized by the recompressing compressor (also referred to as the "recompressor") and reintroduced at a point between the low-temperature and high-temperature recuperators. Because of the rapidly changing properties of carbon dioxide near the critical point, the cold stream of the low-temperature recuperator has a higher specific heat capacity than the hot stream. The bypass loop functions to balance the capacitance rates of these streams by decreasing the mass flow rate through the cold side of the low-temperature recuperator, resulting in higher efficiency

1. Referred to in this dissertation as a “simple cycle”.

compared to the simple recuperated cycle. Note that when no flow is diverted to the recompressor, the recompression cycle becomes functionally equivalent to the simple cycle.

2.1. Turbomachinery

The design-point performance of the compressors and turbine are modeled assuming adiabatic operation with either a constant isentropic efficiency (η_{isen}) or a constant polytropic efficiency (η_{poly}). The polytropic efficiency is defined as the isentropic efficiency over an infinitesimally small pressure ratio and it is used to account for the fact that isentropic efficiency is dependent on the pressure ratio over which it is operating. That is, a turbomachine designed to operate under a higher pressure ratio with a given isentropic efficiency will, typically, be more expensive than a turbomachine with the same efficiency operating under a lower pressure ratio. Another advantage of using polytropic efficiency is that it is independent of the number of stages used by a turbomachine operating across a fixed pressure ratio. The disadvantage of using polytropic efficiency is that it is computationally slower than assuming an isentropic efficiency due to the required integration of the entire compression process over a series of small changes in pressure.

When assuming a constant isentropic efficiency, the specific enthalpy and entropy of the fluid entering the turbomachine inlet (h_{in} and s_{in}, respectively) are determined based on the known temperature and pressure at the inlet. The specific enthalpy at the outlet of the turbomachine that would result if the fluid were isentropically processed to the outlet pressure ($h_{out,i}$) is determined based on the inlet specific entropy and the outlet pressure. The isentropic specific work (w_i) of a turbomachine is calculated according to:

$$w_i = h_{in} - h_{out,i} \tag{2.1}$$

Applying the definition of isentropic efficiency results in an actual specific work (w) for a compressor (left equation) and a turbine (right equation) of:

$$w_{comp} = \frac{w_i}{\eta_{isen}} \qquad w_{turbine} = w_i \eta_{isen} \tag{2.2}$$

The specific enthalpy of the fluid at the outlet of the adiabatic turbomachine (h_{out}) is determined using the following energy balance:

$$h_{out} = h_{in} - w \tag{2.3}$$

This calculated outlet enthalpy and the known outlet pressure fully defines the thermodynamic state at the outlet of the turbomachine.

The constant polytropic efficiency assumption is accounted for in the turbomachinery model by dividing the total pressure ratio into many small pressure differences (200 are currently used in the model) and applying Equations (2.1-2.3) to each incremental change in pressure. The result of this integration process is a final outlet enthalpy (h_{exit}), which is used to calculate an equivalent isentropic efficiency:

$$\eta_{isen} = \frac{h_{out,i} - h_{in}}{h_{exit} - h_{in}} \tag{2.4}$$

2.2. Heat Exchangers

Heat exchangers are modeled assuming a counter-flow configuration with constant conductance (UA). Conductance is preferred over heat exchanger effectiveness because it more directly relates to the size (and cost) of the heat exchangers, while still appropriately characterizing cycle performance under various design conditions. Furthermore, the performance of cycles with different design points can be more directly compared when the total recuperator conductance is fixed. Fixing the value of effectiveness may correspond to dramatically different conductance values (and thus size and cost) for a given heat load under different design conditions and result in misleading conclusions about the relative economic merit of these cycles.

In order to accurately capture the effects of changing carbon dioxide properties, each recuperator is discretized into sub-heat exchangers connected in series (Nellis & Klein, 2012). Using this approach, the total conductance for a given heat exchanger is determined from the inlet conditions and a required rate of total heat transfer ($\dot{Q}$). The total rate of heat transfer for the overall heat exchanger is then evenly divided among the discretized sub-heat exchangers and the application of energy balances on each sub-heat exchanger allows the full determination of the inlet and outlet states for each section. Between 10 and 20 sub-heat exchangers are typically used for the results discussed in this dissertation.

For each sub-heat exchanger, the average capacitance rate ($\dot{C}$) of each stream is calculated according to:

$$\dot{C} = \dot{m}\frac{h_{in} - h_{out}}{T_{in} - T_{out}} \tag{2.5}$$

The effectiveness of each sub-heat exchanger is defined as:

$$\varepsilon = \frac{\dot{Q}_i}{\dot{C}_{min}\left(T_{hot,in} - T_{cold,in}\right)} \tag{2.6}$$

where $\dot{Q}_i$ is the heat transfer in the sub-heat exchanger and $\dot{C}_{min}$ is the minimum of the hot and cold capacitance rates calculated using Eq. (2.5) The dimensionless number of transfer units (*NTU*) for a sub-heat exchanger is calculated for a counter-flow heat exchanger using:

$$NTU = \begin{cases} \dfrac{\log\left(\dfrac{1-\varepsilon C_R}{1-\varepsilon}\right)}{1-C_R} & \text{if} \quad C_R \neq 1 \\ \dfrac{\varepsilon}{1-\varepsilon} & \text{otherwise} \end{cases} \tag{2.7}$$

where C_R is the capacitance rate ratio between the two streams, defined as the ratio of the minimum capacitance rate stream to the maximum capacitance rate stream:

$$C_R = \frac{C_{\min}}{C_{\max}} \tag{2.8}$$

The conductance of each sub-heat exchanger is determined from the *NTU* and minimum capacitance rate:

$$UA_i = NTU\ \dot{C}_{\min} \tag{2.9}$$

The total conductance for the heat exchanger is the sum of the sub-heat exchanger conductance values:

$$UA = \sum_{i=1}^{N_{hxrs}} UA_i \tag{2.10}$$

2.3. Cycle Model

Apart from turbomachinery efficiencies and heat exchanger conductance values, the design-point cycle model inputs (shown in bold in Fig. 2.1) include the main compressor inlet temperature and pressure (also referred to as the "low-side" temperature and pressure), the main compressor outlet pressure (referred to as the "high-side" pressure), the recompression fraction, the turbine inlet (high-side) temperature, and the target net mechanical power output of the cycle (not considering any generator or parasitic losses). Specifying these values fully constrains the set of equations derived from energy and mass balances on the components in the cycle. However, iteration is necessary to solve the set of equations as there is no closed-form solution to the cycle model. A flowchart of the iteration logic used in the system model is shown in Figure 2.2. Because the main compressor and turbine inlet temperatures are specified at the design point, the precooler and primary heat exchanger do not need to be explicitly modeled; the implied assumption is that the heat rejection and heat addition mechanisms are appropriately designed. This assumption reduces the computational complexity of the design-point cycle model and increases its flexibility, as various types of heat rejection or heat addition systems can be modeled separately and coupled to the system model through calculated temperatures and heat rates.

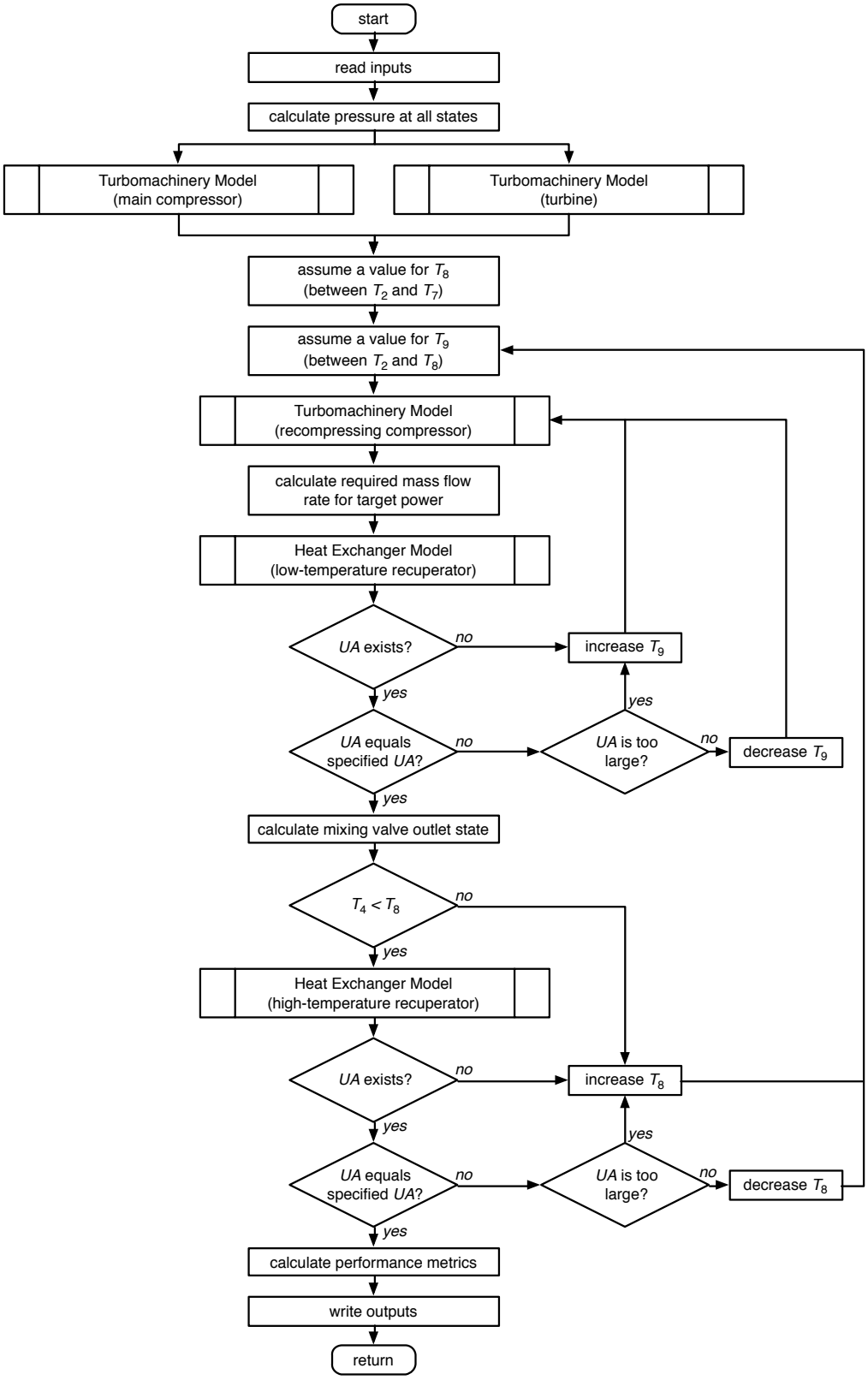


Figure 2.2. Iteration process for the design-point cycle model.

The numerical solution to the design-point model, given the inputs shown in bold in Fig. 2.1, is determined using a nested iteration strategy. The outer iteration loop is initiated by guessing a value for the temperature at state 8 (T_8) that is bounded on the low side by T_2 and on the high side by T_7. Given a value for T_8, control moves to the inner iteration loop, which is initiated by guessing a value for T_9 that is bounded by T_2 and T_8. Setting T_9 establishes the conductance of the low-temperature recuperator, as described in Section 2.2 of this document. Note that if T_9 is equal to T_8, then the heat exchanger conductance is zero; as T_9 decreases the low-temperature recuperator conductance increases to infinity (as the minimum temperature difference between the hot and cold streams goes to zero). The inner loop iterates on T_9 in order to match the calculated and specified low-temperature recuperator conductance to within a given relative tolerance (typically on the order of 10^{-5}). It is possible during the iteration process that the cold-stream temperature within one of the sub-heat exchangers will be warmer than the hot-stream temperature. If this situation occurs it implies that the results are non-physical due to a violation of the Second Law of thermodynamics; non-physical results indicate that the iteration value for T_9 is too low.

Once the inner loop has converged, the recompressor outlet state (10) is determined and the temperature of the working fluid at the outlet of the mixing valve (T_4) is calculated using mass and energy balances; the valve is assumed to be adiabatic. While the inner iteration loop guarantees that T_3 will be less than T_8, the result of this energy balance could lead to a value of T_4 that is larger than T_8. This situation implies there was a Second Law violation in the high-temperature recuperator, and therefore the value for T_8 is too low and the outer iteration loop is cycled with a better estimate for T_8.

If there is no Second Law violation between states 4 and 8, the conductance of the high-temperature recuperator is calculated using T_4, T_7, and T_8. As was the case for the inner iteration loop, the value for T_8 is adjusted until the calculated conductance for the high-temperature recuperator matches the specified conductance to within some relative tolerance. Note that the inner loop is completely evaluated at every iteration of the outer loop.

For both iteration loops, a combination of the bisection and secant methods are used to adjust T_8 or T_9. The secant method has a higher rate of convergence, but sometimes predicts a new temperature that is not physical (outside valid bounds). If this occurs, the model reverts back to the bisection method. A general discussion on both root-finding algorithms is available in Chapra and Canale (2009).

2.4. Fluid Properties

The properties of carbon dioxide change rapidly near the critical point, and accurate fluid property data are required to model SCO_2 Brayton cycles. The two approaches explored in this work effort are 1) using a general equation of state, and 2) using a piecewise interpolated equation of state. Using an equation of state enables high accuracy but is computationally expensive, while the interpolation approach is extremely fast but must be sufficiently refined in order to achieve adequate accuracy.

The National Institute of Standards and Technology (NIST) distributes the Reference Fluid Thermodynamic and Transport Properties Database (REFPROP), a program that is capable of calculating the thermodynamic and transport properties of a large number of fluids and mixtures (E.W. Lemmon et al., 2013). The properties of carbon dioxide are calculated by REFPROP according to the Helmholtz free energy equation of state (Span & Wagner, 1996).

While using REFPROP is extremely accurate, it is computationally expensive. In order to increase the speed of the models, the Fluid Property Interpolation Tables (FIT) software library developed by Northland Numerics (Northland Numerics, 2014) is also implemented. This library uses a piecewise interpolation of Helmholtz free energy and derives all other thermodynamic properties from its analytical derivatives. In order to evaluate the two libraries, as well as providing flexibility for alternative fluid property libraries, the "black box" approach used for the off-design component models (discussed in Chapter 3) is also applied to the calculation of fluid properties. This approach allows the design-point and off-design models to easily switch between various carbon dioxide property libraries and, in fact, enables other fluids besides carbon dioxide to be modeled.

3. Off-Design Modeling Methodology

The current interest in SCO_2 Brayton cycles across a range of applications (solar, nuclear, waste-heat recovery, etc.) provides a unique challenge in designing a modeling methodology that can readily be applied to multiple designs. The developed modeling tool addresses this challenge by assuming "black box" component models. In other words, any models can be used to describe the performance of the turbomachinery and heat exchangers during off-design cycle analysis. The only limitations imposed on the component models are the expected interface; that is, the values required as inputs and those determined as outputs. This black box approach allows a user to easily tailor the system-level cycle model and component models to a specific application. The sections that follow will discuss the required interfaces for the component models, as well as the currently implemented off-design component models.

3.1. Component Model Interfaces

In order to increase the applicability of the developed modeling tool, any models that satisfy a few criteria can be used to represent the off-design performance of the individual components that comprise the cycle. This flexibility is accomplished by defining a number of interfaces for the component models; any models that can be implemented with these interfaces are suitable. Each of the off-design turbomachinery models have related but unique requirements; the inputs that each may use and the outputs each must provide are listed in Tables 3.1-3.3. The differences between the three sets of requirements for the turbomachinery models are due to their respective roles in the off-design cycle model, which are discussed in Section 3.5. Note that the "Design Parameters" input is actually a number of variables; the method by which these variables are passed to the model is implementation-specific and is discussed in Chapter 4. Also discussed in Chapter 4 is the interface used to set the Design Parameters of the turbomachinery based on a given design point.

Table 3.1. Required inputs and outputs for compatible off-design compressor models.

Inputs	Outputs
Design Parameters (rotor diameter, design-point shaft speed, etc.)	Outlet Temperature
Inlet Temperature	Outlet Pressure
Inlet Pressure	Specific Work
Mass Flow Rate	Warnings (surge conditions, speed of sound concerns, etc.)
Shaft Speed	

Table 3.2. Required inputs and outputs for compatible off-design recompressor models.

Inputs	Outputs
Design Parameters (rotor diameter, design-point shaft speed, etc.)	Outlet Temperature
Inlet Temperature	Shaft Speed
Inlet Pressure	Specific Work
Mass Flow Rate	Warnings (surge conditions, speed of sound concerns, etc.)
Outlet Pressure	

Table 3.3. Required inputs and outputs for compatible off-design turbine models.

Inputs	Outputs
Design Parameters (rotor diameter, design-point shaft speed, etc.)	Outlet Temperature
Inlet Temperature	Mass Flow Rate
Inlet Pressure	Specific Work
Shaft Speed	Warnings (speed of sound concerns, etc.)
Outlet Pressure	

While heat exchanger conductance values are used to model the recuperators in the cycle, these values can be adjusted under off-design conditions. Similarly, the pressure drop for both fluid streams can be scaled. The purpose of the off-design heat exchanger model in the cycle is to provide these values, and its required inputs and outputs are listed in Table 3.4

Table 3.4. Required inputs and outputs for compatible off-design heat exchanger models.

Inputs	**Outputs**
Design Parameters (design-point conductance and pressure drops, etc.)	Scaled Conductance (*UA*)
Cold Stream Mass Flow Rate	Scaled Cold Stream Pressure Drop
Hot Stream Mass Flow Rate	Scaled Hot Stream Pressure Drop

The off-design heat exchanger model interface does not currently require inlet temperatures or pressures as inputs, as initial model development proceeded under the assumption that, for the recuperators, the effects due to off-design inlet temperature and pressure are secondary to the effects due to off-design mass flow rates. This approach greatly reduces the computational complexity of the off-design cycle model. However for more advanced off-design heat exchanger models, the inlet conditions can be provided at the cost of requiring an iterative approach to determining the conductance and pressure drop values. An iterative approach is necessary if the secondary effects due to off-design inlet temperature and pressure are considered because the calculated conductance will affect the inlet conditions of the recuperator, which in turn affect the calculated conductance. The current approach does not require iteration because the mass flow rate in the cycle is not affected by the thermal performance of the recuperators.

3.2. Sandia National Laboratory SCO_2 Compressor

To facilitate development of the off-design cycle model, the compressor and recompressor models are implemented as a functional relationship between the dimensionless flow and ideal head coefficients of the radial compressor that is currently being developed for use with carbon dioxide at Sandia National Laboratory (Conboy et al., 2012; Wright et al., 2010).

3.2.1 Functional Relationship

The flow coefficient (ϕ) of a compressor is a dimensionless flow rate that is defined according to:

$$\phi = \frac{\dot{m}}{\rho U D^2} \tag{3.1}$$

where $\dot{m}$ is the mass flow rate, ρ is the fluid density at the compressor inlet, D is the diameter of the rotor in the compressor (also called the "compressor diameter"), and U is the rotor tip speed:

$$U = \frac{D}{2} N \tag{3.2}$$

with N representing the shaft speed expressed in radians per second. The ideal head coefficient is a dimensionless pressure rise through the compressor and it is defined according to:

$$\psi_i = \frac{\Delta h_i}{U^2} \tag{3.3}$$

where Δh_i is the isentropic specific enthalpy rise through the compressor (i.e., the specific enthalpy rise that would be obtained if the compressor were isentropic and adiabatic). The flow and head coefficients are derived by applying the Buckingham Pi theorem and are commonly used to describe compressor performance. A more thorough description is available in Peng (2008).

The ideal head coefficient and the compressor efficiency are both functions of the flow coefficient. The variation of head coefficient and isentropic efficiency with flow coefficient is a unique characteristic of each compressor design. The functional relation used within the compressor model is based on empirical data, typically provided by the manufacturer of the compressor in the form of a "performance

map" that is generated using either modeled or experimental data. Figure 3.1 is the performance map for the SNL compressor overlaid onto experimental data. Note that SNL was able to run the compressor beyond the predicted surge line, indicating the model is slightly conservative in regards to the operating range.

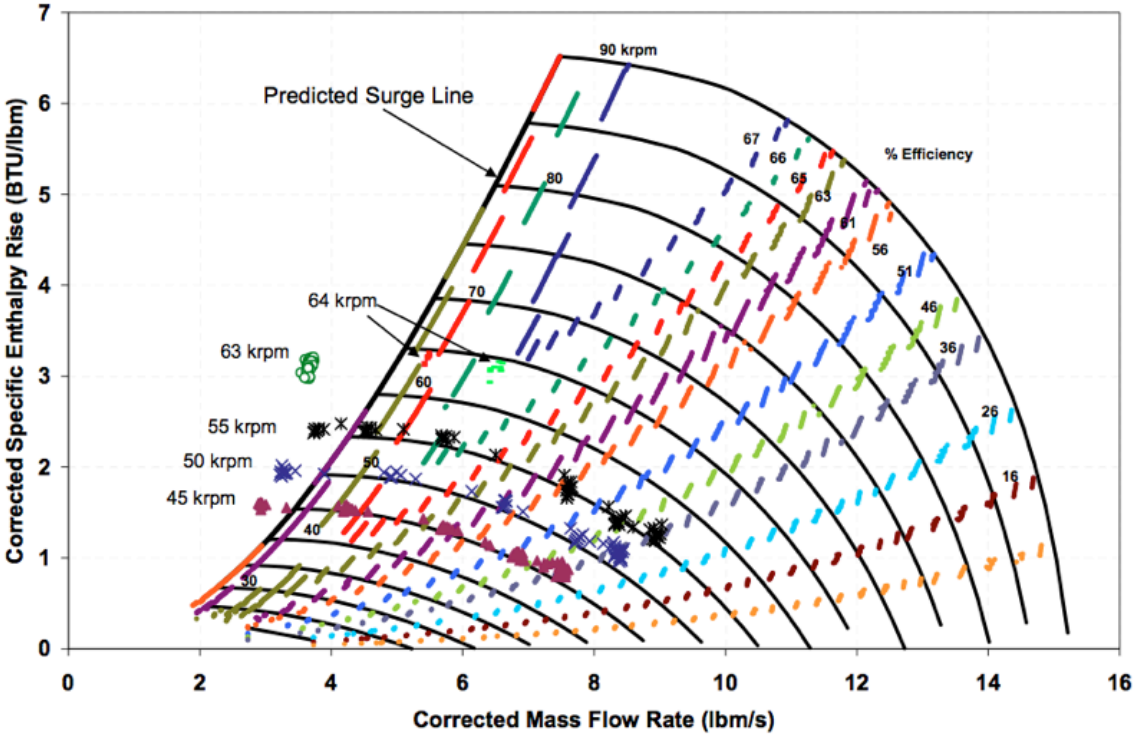


Figure 3.1. Performance map for the main compressor in the Sandia National Laboratory supercritical CO_2 test loop, taken from Wright et al. (2010). Note: the y-axis label should be "Corrected Specific Ideal Enthalpy Rise (BTU/lbm)".

The data used to create the performance map were generated by Barber-Nichols (the compressor manufacturer) using in-house modeling software. Using these modeled data, the ideal enthalpy rise and efficiency for various shaft speeds and mass flow rates were non-dimensionalized using the previously discussed flow (ϕ) and ideal head (ψ_i) coefficients, defined in Equations (3.1) and (3.3), respectively. Using a non-dimensional head-flow curve reduces the number of independent variables needed to calculate compressor efficiency and enthalpy rise from many (shaft speed, input conditions, and mass flow rate) to one (flow coefficient). More importantly, the compressor map shown in Fig. 3.1 is only valid at the com-

pressor inlet conditions for which it was generated, which are 1,115 psi (7,688 kPa) and 549.7°R (32.2°C). For other inlet conditions, the mass flow rate, shaft speed, and ideal enthalpy rise must be corrected using a number of transformations that are derived from dynamic similarity and perfect gas assumptions that also take into account deviations from ideal gas behavior using a compressibility factor. A non-dimensional head-flow curve, however, accounts for deviations from the reference conditions implicitly and no additional corrections are required. To illustrate the advantage of this approach, the following equations are used to transform actual conditions (indicated by the subscript 'a') to the equivalent reference conditions (indicated by the subscript 'eq'), with the reference conditions indicated by the subscript 'ref' (Noall & Forsha, 2008).

$$\dot{m}_{eq} = \frac{\dot{m}_a \sqrt{\theta_1}}{\delta} \varepsilon \tag{3.4}$$

$$N_{eq} = \frac{N_a}{\sqrt{\theta_1}} \tag{3.5}$$

$$\Delta h_{eq} = \frac{\Delta h_a}{\theta_1} \theta_2 \tag{3.6}$$

where,

$$\varepsilon = \frac{\gamma_{ref} \left(\frac{2}{\gamma_{ref} + 1} \right)^{\frac{\gamma_{ref}}{\gamma_{ref} - 1}}}{\gamma_a \left(\frac{2}{\gamma_a + 1} \right)^{\frac{\gamma_a}{\gamma_a - 1}}} \tag{3.7}$$

$$\theta_1 = \left(\frac{V_{cr,a}}{V_{cr,ref}} \right)^2 \tag{3.8}$$

$$\theta_2 = \frac{T_{o,a}}{T_{o,ref}} \tag{3.9}$$

$$\delta = \frac{P_{o,a}}{P_{o,ref}} \tag{3.10}$$

$$V_{cr} = \sqrt{\frac{2\gamma}{\gamma + 1} g_c Z R T_o} \tag{3.11}$$

While Equations (3.4-3.11) are useful for directly adjusting a compressor map for off-design conditions, they are not required when using the non-dimensional head-flow curve approach adopted for this work effort.

Figure 3.2 is a plot of the compressor performance map data after being non-dimensionalized using Eqs. (3.1) and (3.3).

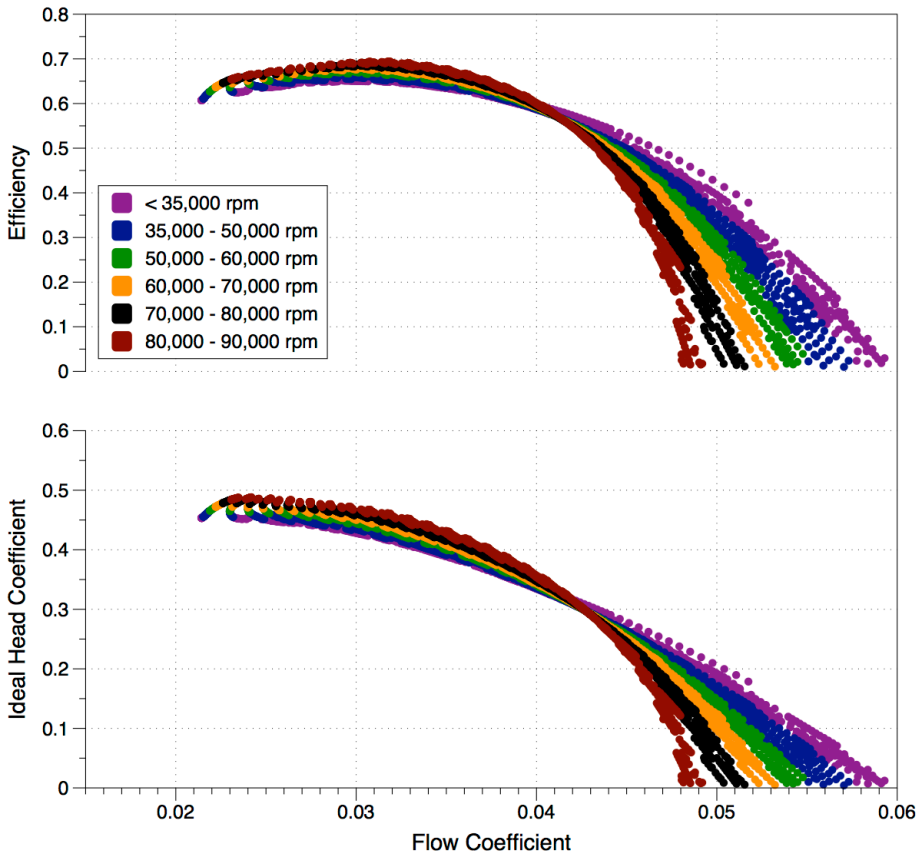


Figure 3.2. Non-dimensional head-flow and efficiency-flow curves for the main compressor in the Sandia National Laboratory supercritical CO_2 test loop; points are colored by shaft speed.

While Fig. 3.2 is clearly a more collapsed representation of the performance data shown in Fig. 3.1, there appears to be an additional dependence on shaft speed that is unaccounted for and causing substantial scatter, especially for larger values of flow coefficient. In order to address this deviation, a number of modifications to the defined relationships for the flow and ideal head coefficients were considered. The

following modified definitions, which are completely empirical and specific to this compressor, were ultimately chosen:

$$\phi^* = \frac{\dot{m}}{\rho U D^2}\left(\frac{N}{N_{design}}\right)^{1/5} \tag{3.12}$$

$$\psi_i^* = \frac{\Delta h_i}{U^2}\left(\frac{N_{design}}{N}\right)^{\left(20\phi^*\right)^3} \tag{3.13}$$

$$\eta^* = \eta\left(\frac{N_{design}}{N}\right)^{\left(20\phi^*\right)^5} \tag{3.14}$$

where N is the shaft speed in rev/min and N_{design} is the shaft speed at the design point (75,000 rev/min). Applying these modified definitions to the raw data for shaft speeds greater than 35,000 rev/min (speeds below which are atypical for the SNL compressor, due to its foil bearings) results in the curves shown in Figure 3.3.

As shown by Fig. 3.3, applying Eqs. (3.12-3.14) collapses the compressor performance map to two curves that are easily fit with fourth order polynomials; these curves are overlaid on the performance map data in Figure 3.4.

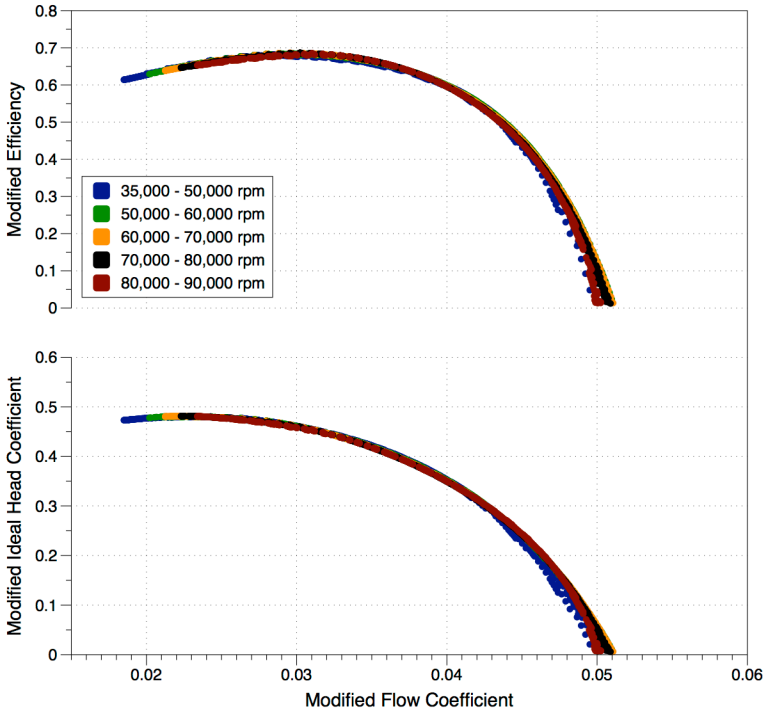


Figure 3.3. Modified non-dimensional head-flow and efficiency-flow curves for the main compressor in the SNL supercritical CO_2 test loop.

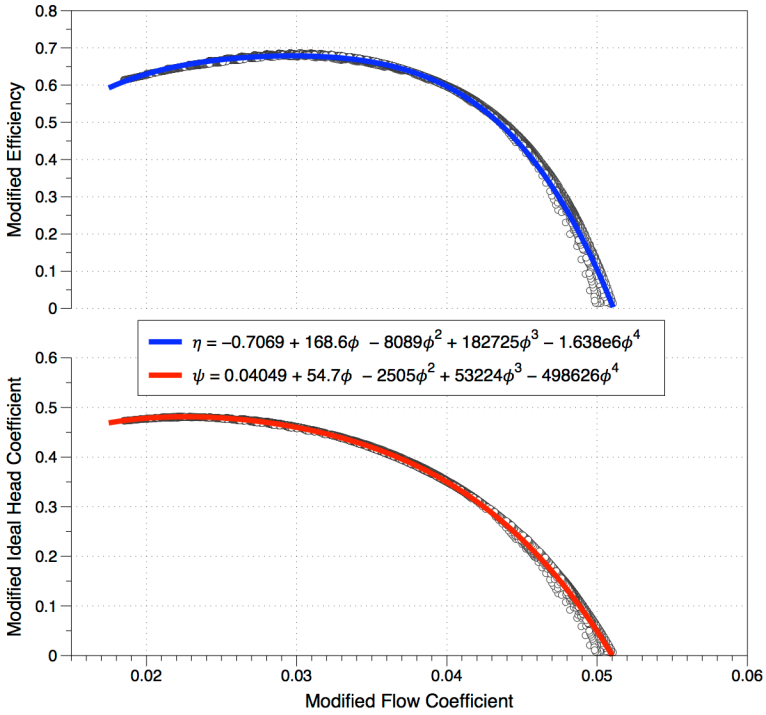


Figure 3.4. Polynomial fits of the modified non-dimensional head-flow and efficiency-flow curves for the main compressor in the SNL supercritical CO_2 test loop.

For this particular compressor, the lower limit for the flow coefficient is roughly 0.02 and operation below this condition will lead to surge, which is an undesirable condition characterized by aerodynamic instability and rapid flow reversals. The exact surge limit is dependent on many variables and must be determined (or confirmed) experimentally for a given compressor. For the purposes of this work effort, constraining the minimum allowable flow coefficient to 0.02 is sufficient. The upper limit of the flow coefficient for compressors similar to this compressor corresponds to an ideal head coefficient of zero, which occurs near a flow coefficient of 0.05. For other compressor designs the upper limit of the flow coefficient may instead be limited by the occurrence of choking within the turbomachine.

The error associated with fitting the dimensionless head-flow and efficiency-flow curves with polynomials is minimal. For the modified ideal head relationship, the standard deviation is 0.011 with an R^2 value of 0.992, and for the modified efficiency relationship, the standard deviation is 0.020 with and R^2 value of 0.983. Figure 3.5 shows the error between the ideal enthalpy rise and efficiency obtained from the curve fits and the modified dimensionless coefficients versus the values provided by the compressor performance map.

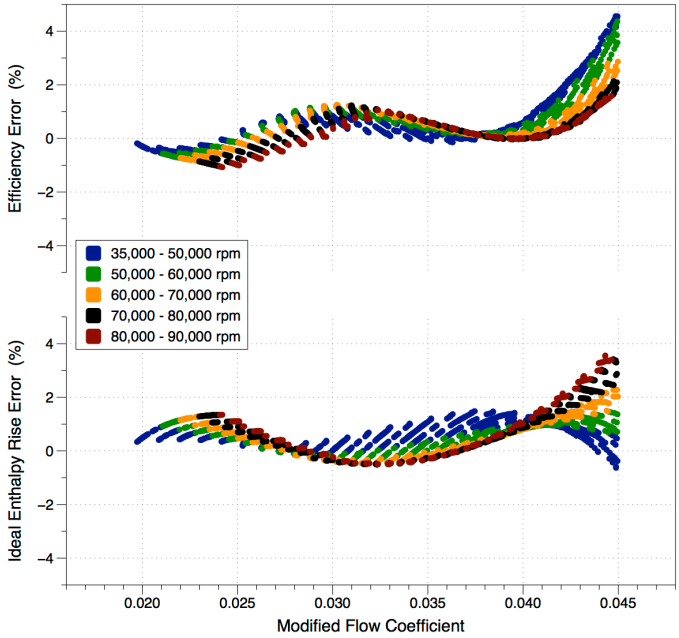


Figure 3.5. Deviations in efficiency and ideal enthalpy rise associated with using the modified dimensionless numbers and polynomial fits.

The results of the analysis indicate that the performance map for the main compressor being studied at SNL can effectively be reduced to a dimensionless head-flow curve and a dimensionless efficiency-flow curve, both of which can be approximated with fourth order polynomials as a function of modified flow coefficient. The non-dimensional head and efficiency curves derived from the main compressor wheel under investigation at SNL are very similar to non-dimensional curves derived (using a similar approach) from the recompressor wheel that is also under investigation at SNL. The recompressor wheel is slightly larger than the main compressor wheel, but the non-dimensional performance is similar with the exception of a reduction in the range of valid flow coefficients. For consistency, the polynomials shown in Fig. 3.4 are used to represent the off-design performance for both the compressor and recompressor in the currently implemented models. Note that the polynomials are only valid for flow coefficient values greater than 0.02 (corresponding to the predicted surge region). The polynomials are also limited to positive values of ideal head coefficient and efficiency.

3.2.2 Experimental Validation

While it was shown that the compressor map can be effectively simplified using dimensionless coefficients, the previous analysis does not account for deviations from the reference conditions of the performance map. In order to show that Eqs. (3.4-3.11) can be omitted when using a dimensionless approach, experimental data are required to validate the aforementioned methodology.

Sandia National Laboratory provided experimental data for a number of compression tests at various compressor inlet conditions. These conditions are summarized in Table 3.5.

Table 3.5. Inlet conditions for various carbon dioxide compression tests.

Test Case Name	Average Inlet Pressure (kPa)	Average Inlet Temperature (°C)
Map Reference	7,540	32
Supercritical	9,105	35
Liquid	7,219	28
Gas	6,771	28

For each test case, the mass flow rate and fluid density at the compressor inlet are measured with high accuracy (on the order of 0.1%) using a Coriolis flow meter, pressures are measured with transducers accurate to ± 34 kPa (± 5 psi), and temperature is measured using thermocouples accurate to ± 1°C. More information about the experimental setup is available in Conboy (2012). The properties of carbon dioxide at the compressor inlet are calculated using pressure and density, and the ideal enthalpy rise through the compressor is calculated using the entropy at the compressor inlet and the pressure at the compressor outlet. For this analysis, fluid properties are provided by the Engineering Equation Solver (EES) software package (F-Chart Software, 2013). The shaft speed was held constant for each test at a number of speeds and the mass flow rate through the loop was varied using a valve to provide increased or decreased flow resistance in the loop. Figure 3.6 is a plot of the recorded data for a portion of the Map Reference test case. Plots of the recorded measurement data for the four test cases are available in Appendix VIII.

For the Map Reference case, the shaft speed was held constant at 45,000 rpm and 50,000 rpm and the mass flow rate through the loop was incremented four and five times, respectively. At each increment of mass flow rate the pressure rise through the compressor achieves a steady state condition, shown as the red lines in Figure 3.7.

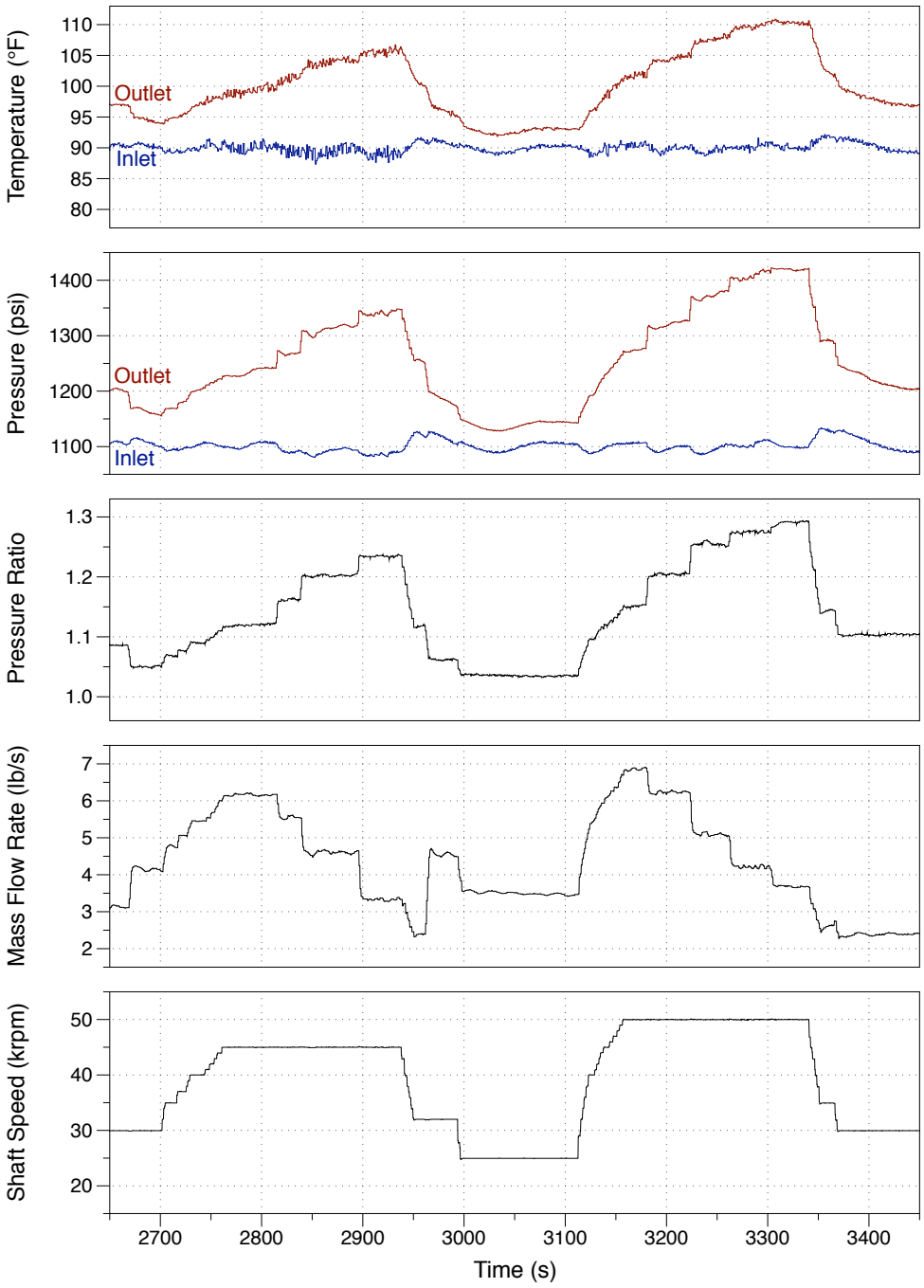


Figure 3.6. Recorded measurement data for the Map Reference test case.

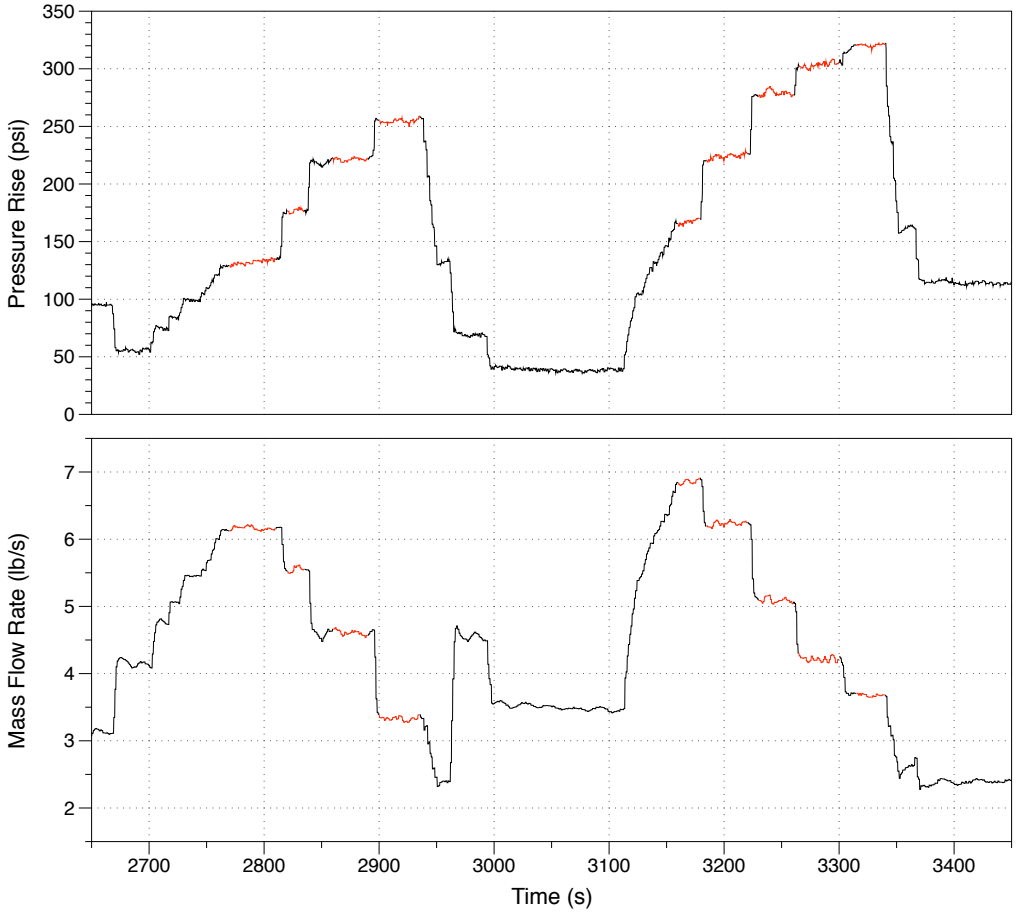


Figure 3.7. Recorded mass flow rate and pressure rise through the compressor for the Map Reference test case where the regions highlighted in red are assumed as steady-state operation.

The modified flow and ideal head coefficients were calculated for each of the steady state regions (highlighted in red in Fig. 3.7) by averaging the recorded data and applying Eqs. (3.12) and (3.13). The modified flow coefficient was used to predict the modified ideal head coefficient using the polynomial fit discussed above. This approach was repeated for the other three test cases, and the measured and predicted modified ideal head coefficients are compared in Figure 3.8. The error bars in Fig. 3.8 are calculated using uncertainty propagation in EES and are based on the larger of the measurement error or standard deviation for each steady state region.

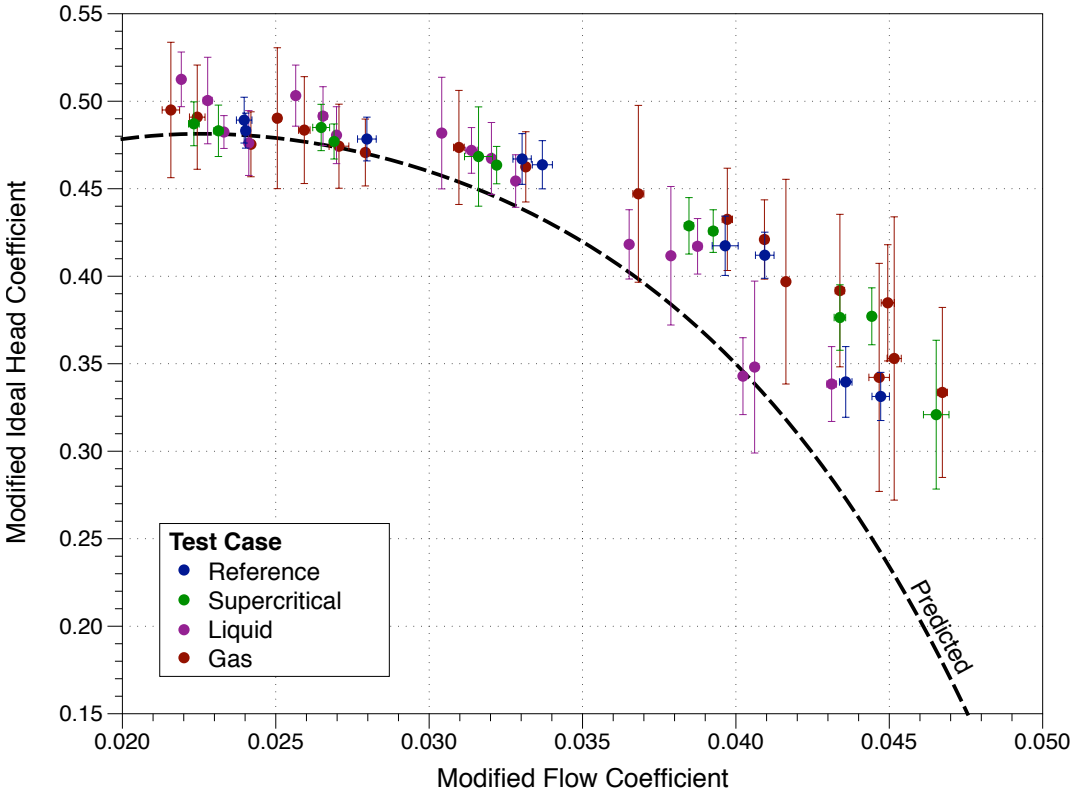


Figure 3.8. Measured and predicted modified ideal head coefficient for the four test cases.

The measured relationship between flow and ideal head coefficients agrees well with the relationship predicted by the non-dimensional performance curve for various inlet conditions, without having to correct for deviations from reference conditions. At larger flow coefficients the measured head coefficient does not decrease as rapidly as predicted; this discrepancy was also observed by researchers at SNL when using the compressor map with corrected inlet conditions. The likely cause of this discrepancy is that the map originally provided by the manufacturer does not accurately capture the performance of this compressor at larger mass flows and lower pressure ratios. While the current relationship between the flow and ideal head coefficients derived from the performance map is satisfactory, a new relationship could be generated from the data shown in Fig. 3.8 and validated with future experimental results. Alternative compressor hardware can be characterized using the same technique, provided data were available.

3.3. Low-Reaction Radial Turbine

For the purposes of generating and testing the framework developed for this project, the initial turbine model development is applicable to radial designs, which are appropriate for SCO_2 applications up to 50 MWe (Gibbs et al., 2006). The mass flow rate through a radial turbine is strongly dependent on inlet conditions and outlet pressure and, depending on its blade and nozzle design, weakly dependent on shaft speed. Modeling the turbine as a constant area un-choked nozzle, this relationship is:

$$\dot{m} = C_s \, A_{nozzle} \, \rho \tag{3.15}$$

where ρ is the outlet density of the fluid and A_{nozzle} is the effective nozzle area that results in the correct relationship between mass flow rate and spouting velocity (C_s) and is based on the geometry of the turbine. The spouting velocity is the velocity that would be achieved if the fluid were expanded isentropically to the outlet pressure in an ideal nozzle and is calculated by:

$$C_s = \sqrt{2\Delta h_i} \tag{3.16}$$

where Δh_i is the change in specific enthalpy across the turbine assuming an isentropic process. Modeling the turbine as a constant area un-choked nozzle assumes a radial inflow turbine with a low degree of reaction; a low degree of reaction (also referred to as a reaction ratio) means that the majority of the isentropic enthalpy change is through the nozzles of the turbine, as opposed to along the blades.

An important characteristic of radial turbine performance is the ratio of tip speed (U) to spouting velocity, also called the velocity ratio:

$$\nu = \frac{U}{C_s} \tag{3.17}$$

Chen and Baines (1994) have proposed a general relationship between the ideal efficiency (η_{ideal}) and velocity ratio of a radial turbine that takes into account blade geometry and the loading. For a well-designed turbine with a low loading coefficient, the relationship simplifies to:

$$\eta_{ideal} = 2\nu\sqrt{1-\nu^2} \tag{3.18}$$

Plotting Eq. (3.18) results in the familiar relationship between ideal efficiency and velocity ratio for a radial turbine, shown in Figure 3.9. Note that the maximum efficiency occurs at a velocity ratio of 0.707, as is expected based on Japikse and Baines (1997).

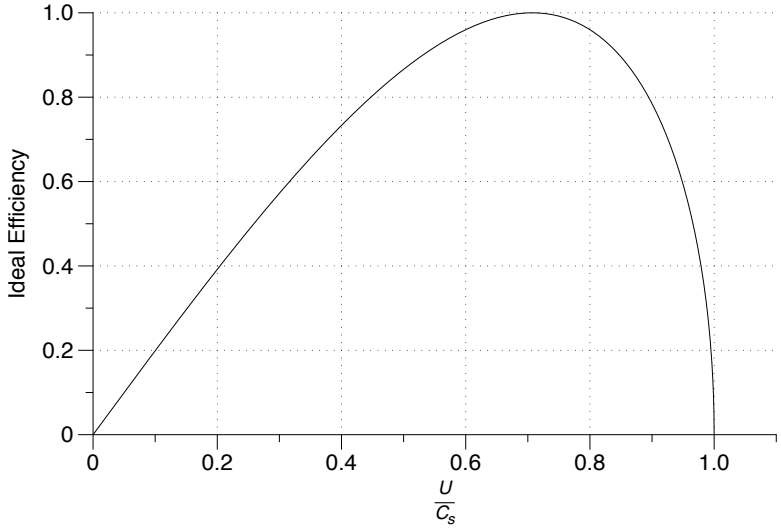


Figure 3.9. Efficiency of an ideal radial turbine as a function of velocity ratio.

The ideal efficiency does not take into account other losses associated with the turbine (e.g., windage losses). To account for these losses in the semi-empirical model the efficiency of the turbine (η) is calculated by scaling the ideal efficiency by the turbine design point efficiency (η_{design}):

$$\eta = \eta_{design}\,\eta_{ideal} = \eta_{design}\,2\nu\sqrt{1-\nu^2} \tag{3.19}$$

Alternative turbine configurations suitable for use in an SCO_2 power cycle are currently being evaluated and tested at Sandia National Laboratory (Wright et al., 2011). The manufacturer of the turbine wheels, Barber-Nichols, provided performance maps such as the one shown in Figure 3.10. These maps can be used to predict turbine performance under off-design conditions in a similar manner to the compressor performance maps discussed earlier (i.e., correcting the inlet conditions using various scaling equations). Alternatively, the performance map information can be used to test the proposed low-reaction turbine model.

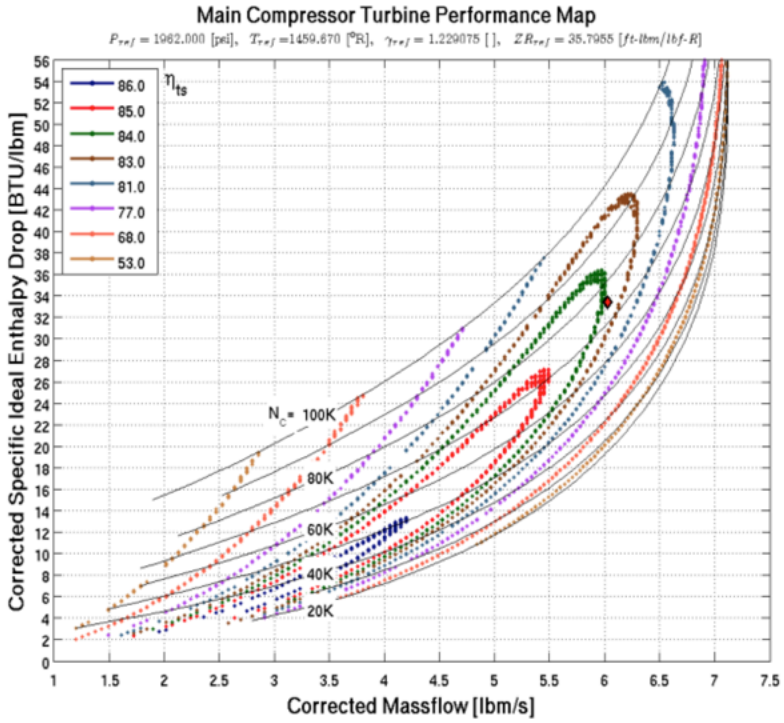


Figure 3.10. Performance map for a turbine used in the SCO_2 test loop at Sandia National Laboratories, taken from the presentation associated with Wright et al. (2011).

The first order approximation described by Eq. (3.15) is tested using the performance map in Fig. 3.10 by calculating the effective nozzle area from the mass flow rate and ideal enthalpy drop information that is provided by Barber-Nichols. It should be noted that this information does not correspond to actual experimental test data; rather, this information was generated by Barber-Nichols using proprietary in-house modeling tools. Figure 3.11 shows the result of applying Eqs. (3.15) and (3.16) to the performance map data for the main shaft turbine.

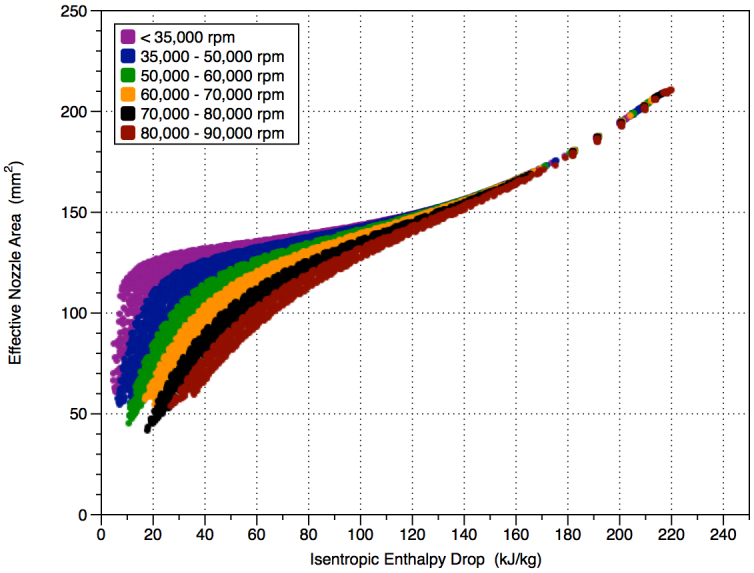


Figure 3.11. Calculated effective nozzle area using data from the turbine performance map; the markers are colored by shaft speed.

The implemented radial turbine model assumes that the effective nozzle area remains constant under off-design conditions. The application of Eq. (3.15) to the performance map data shows that both shaft speed and isentropic enthalpy drop (which is related to pressure drop) have an effect on the effective nozzle area for the turbine developed by Barber-Nichols. Furthermore, the shaft speed has a larger effect at lower pressure drops, implying that the degree of reaction for the SNL turbine varies under off-design conditions. Interestingly, using the inlet density of the turbine in Eq. (3.15) results in a more constant calculated effective nozzle area, shown in Figure 3.12. These results indicate that the SNL turbine has a non-negligible degree of reaction. That is, not all the pressure drop across the turbine is through the nozzles. While the assumed low-reaction radial model does not fit the data of the SNL turbine as neatly as the compressor, the flexibility of the modeling methodology allows both of these variations to be implemented and compared to determine their effect on system-level performance, as discussed in Chapter 6.

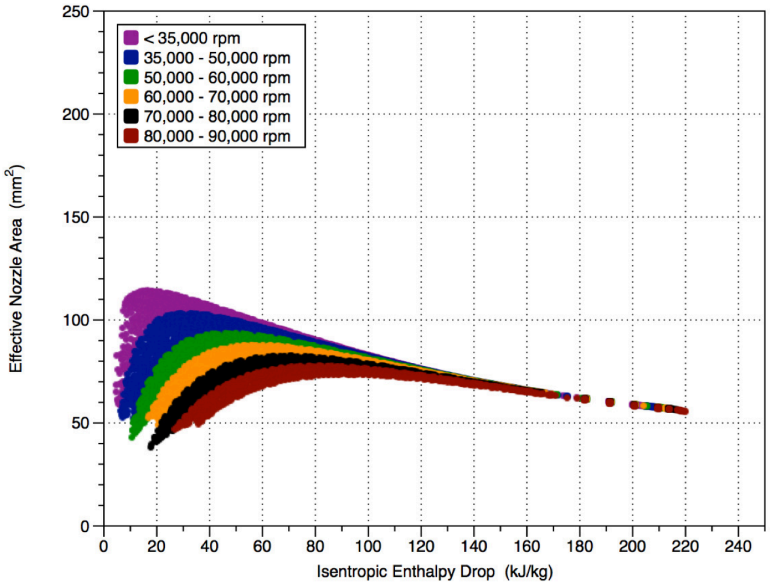


Figure 3.12. Calculated effective nozzle area using data from the turbine performance map and inlet fluid density; the markers are colored by shaft speed.

The performance map provided by Barber-Nichols can also be compared to the efficiency calculated using Eq. (3.19), as shown in Figure 3.13, which is a plot of the efficiency predicted by the performance map as a function of ν overlaid by the efficiency predicted from Eq. (3.19) and scaled with an appropriate value of η_{design}.

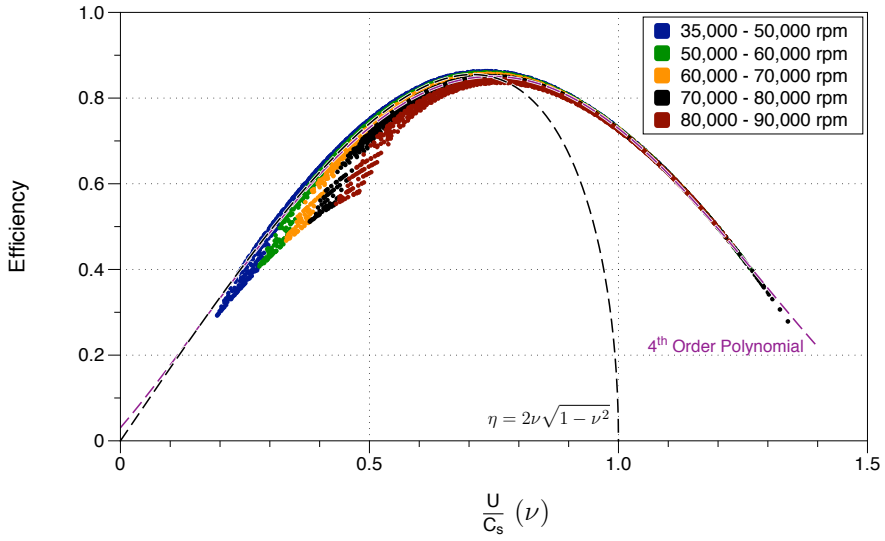


Figure 3.13. Efficiency predicted from the performance map as a function of the ratio of tip speed to spouting velocity.

The performance map data agree well with Eq. (3.19) as the value of ν increases to 0.7, which corresponds roughly to the maximum operating efficiency of the turbine. Beyond a ν of 0.7, though, the performance map and Eq. (3.19) begin to diverge. The values for ν that are greater than one are another indication that the SNL turbine has a non-negligible degree of reaction. However, the current low-reaction turbine model under-predicts the efficiency at larger values of ν, so results based on the simplified model will be conservative. For higher-performance predictions, the efficiency curve based on the 4th order polynomial fit plotted as the purple dashed line in Fig. 3.13 is implemented in an alternative off-design turbine model based more closely on the SNL turbine. This alternative model uses the inlet fluid density for Eq. (3.15) when predicting the allowable mass flow rate through the turbine.

3.4. Heat Exchangers

The performance of the recuperators in the off-design cycle model is represented using specified conductance values and pressure drops. However, under off-design mass flow rates the conductance and pressure drops through the heat exchangers will differ from their design values. To account for this deviation, an off-design heat exchanger model is implemented based on scaling pressure drop and conductance with mass flow rate. Thermal capacitance effects are not considered while modeling the off-design performance of the heat exchangers.

A common method for determining the pressure drop due to friction for internal flow is the use of the Darcy friction factor (f), a dimensionless number (Fox et al., 2008) that is defined such that:

$$\Delta P = f \frac{L}{D} \frac{\rho V^2}{2} \tag{3.20}$$

For a given heat exchanger, the length (L) and hydraulic diameter (D) are constant, and it follows that ΔP is proportional to the friction factor (f), fluid density (ρ), and fluid velocity (V) to the second power:

$$\Delta P \propto f\ \rho\ V^2 \tag{3.21}$$

The velocity of the fluid in the heat exchanger can be expressed in terms of the mass flow rate:

$$V = \frac{\dot{m}}{\rho A_c} \tag{3.22}$$

Substituting Eq. (3.21) into Eq. (3.22) and recognizing that the cross sectional area of the flow passage in the heat exchanger (A_c) is constant results in:

$$\Delta P \propto f \rho^{-1} \dot{m}^2 \tag{3.23}$$

There are a number of different correlations available for calculating the friction factor (f), but for the purpose of scaling pressure drop with mass flow rate, the simple Blasius correlation (Fox et al., 2008) is used:

$$f = \frac{0.316}{\mathrm{Re}^{1/4}} \tag{3.24}$$

where the Reynolds Number (Re) is defined as:

$$\mathrm{Re} = \frac{\rho V D_H}{\mu} \tag{3.25}$$

The hydraulic diameter (D_H) is a constant for a given heat exchanger, so combining Eqs. (3.22), (3.24), and (3.25) results in the friction factor scaling with mass flow rate and fluid viscosity according to:

$$f \propto \dot{m}^{-1/4} \mu^{1/4} \tag{3.26}$$

Combining Eqs. (3.23) and (3.26) results in:

$$\Delta P \propto \dot{m}^{7/4} \, \mu^{1/4} \, \rho^{-1} \tag{3.27}$$

Assuming that the average fluid properties within the heat exchanger do not change significantly in off-design conditions, the pressure drop through a heat exchanger scales with mass flow rate according to:

$$\Delta P = \Delta P_{design} \left(\frac{\dot{m}}{\dot{m}_{design}} \right)^{7/4} \tag{3.28}$$

where ΔP_{design} and $\dot{m}$ are the pressure drop and mass flow rate associated with the design point.

A similar analysis for heat exchanger conductance using the Dittus-Boelter correlation is presented in Patnode (2006) and results in:

$$UA = UA_{design}\left(\frac{\dot{m}}{\dot{m}_{design}}\right)^{0.8} \tag{3.29}$$

The assumption that the viscosity and density do not change significantly in the heat exchanger under off-design conditions is not valid in the vicinity of the critical point of carbon dioxide. However, the relationships presented in this section are applied only to the recuperators in the cycle, which typically operate at higher temperatures away from the critical point. Depending on the type of analysis being performed and the range of off-design conditions being considered, it may be necessary to implement more detailed scaling relations.

3.5. Cycle Model

The inputs and outputs of the off-design compressor and turbine models (Tables 3.1 and 3.3, respectively) are complimentary in that the main compressor model uses the mass flow rate as an input and returns the compressor outlet pressure (i.e., there is a head-flow curve that relates pressure rise to mass flow rate), while the turbine model uses the inlet conditions and outlet pressure as inputs and returns the mass flow rate (i.e., there is a flow resistance afforded by the fixed area restriction in the turbine). Matching the head-flow curve of the compressor with the flow resistance of the turbine, as well as flow resistance associated with heat exchanger and piping pressure drops, allows the operating point of the system to be determined. Figure 3.14 illustrates this concept, showing the effects of changing shaft speed and turbine inlet temperature on the operating point of a simple recuperated Brayton cycle. Note that these results are intended only to show the general trends of the relationship between compressor and turbine operation.

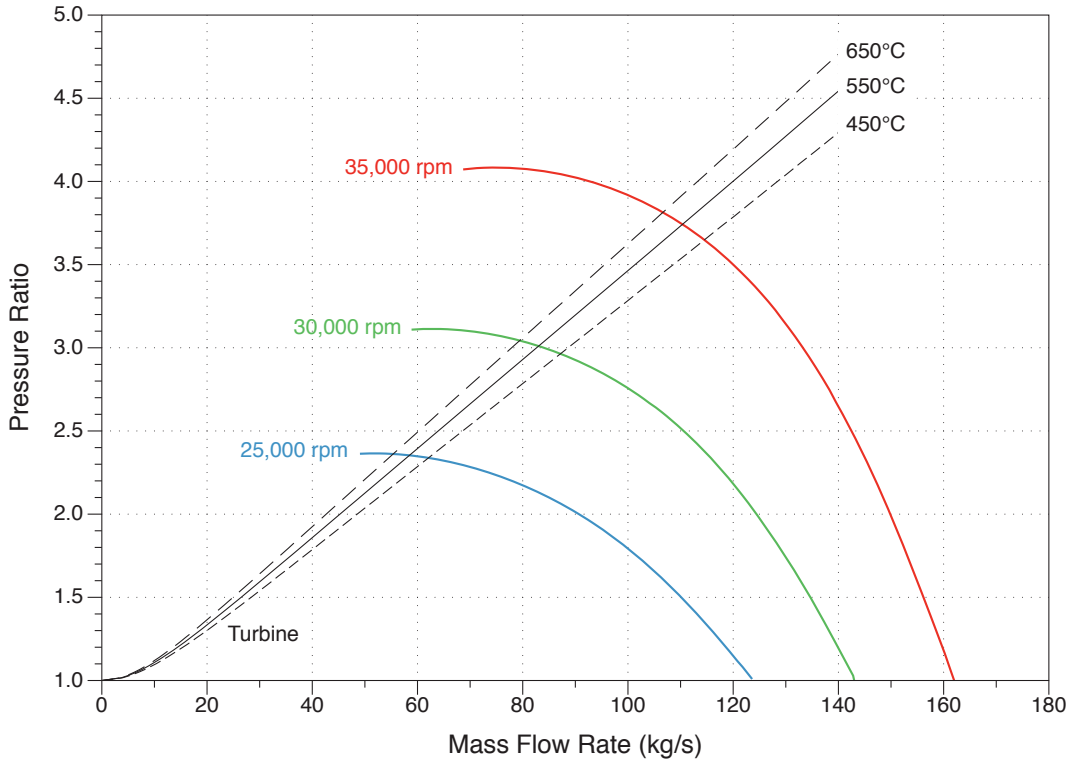


Figure 3.14. Head-flow and flow resistance curves for different shaft speeds and turbine inlet temperature; the curves intersect at the corresponding operating point of the cycle.

The intersection points of the compressor and turbine curves correspond to the operating point of the system. Note that changing the speed of the compressor has a significant impact on the operating point, whereas a change in turbine inlet temperature has a much smaller effect. Figure 3.14 is meant to illustrate the concept of matching compressor and turbine curves and a number of simplifying assumptions were made such as no pressure drops in the heat exchangers.

Before discussing the iteration strategy used to determine the operating point of the cycle under off-design conditions, it is useful to give a brief overview of the off-design model inputs. The inputs, shown in bold in Figure 3.15, are the main compressor inlet temperature ($T_{mc,in}$) and pressure ($P_{mc,in}$), the turbine inlet temperature ($T_{t,in}$), the recompression fraction (ϕ_{rc}), and the shaft speeds of the main compressor (N_{mc}) and turbine (N_t).

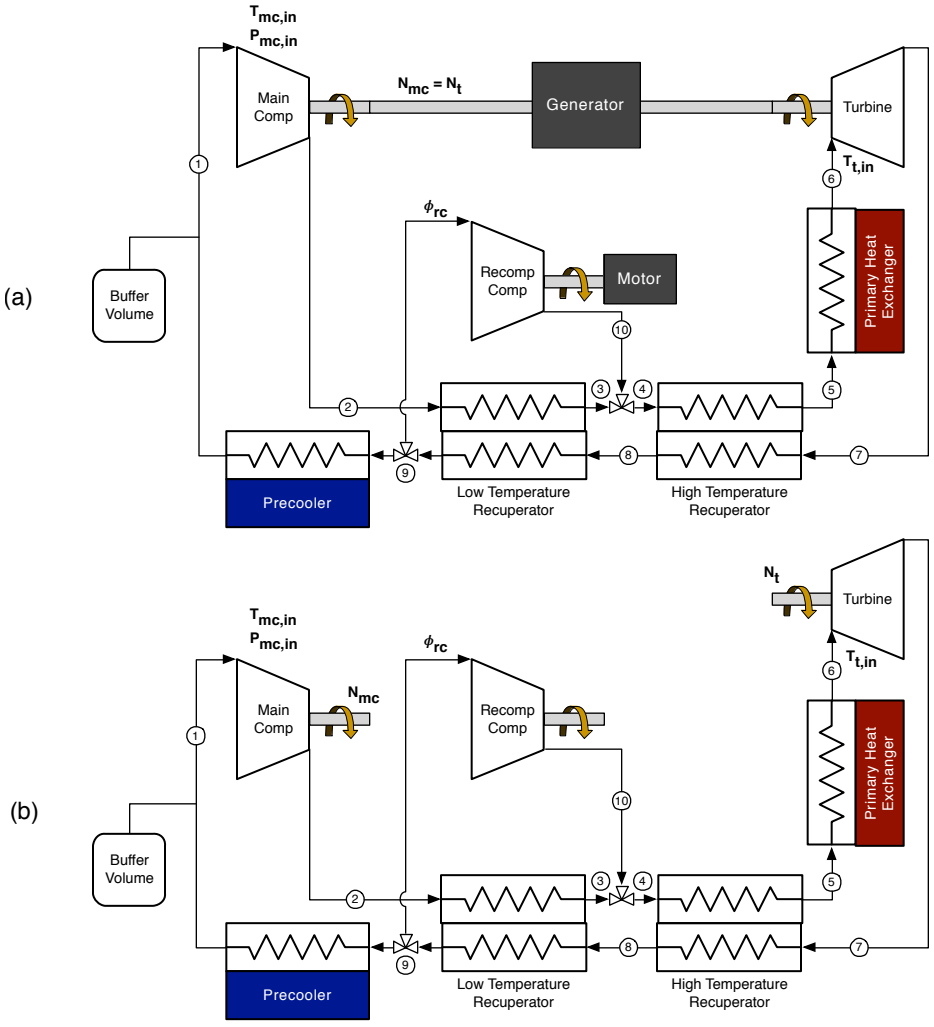


Figure 3.15. Diagram of a single-shaft recompression Brayton cycle (a) and a split-shaft recompression Brayton cycle (b); model inputs are shown in bold.

The split-shaft configuration in Fig. 3.15(b) is characterized by the use of separate shafts for the main compressor and the turbine. (Not pictured are the motors for each compressor and the generator associated with the turbine). A split-shaft configuration is advantageous in that it allows for a constant-speed, synchronous generator tied directly to the electrical grid, but it is more complex than a single-shaft configuration and the additional motor required to drive the main compressor will introduce an additional inefficiency in the system. In the interest of maximizing the flexibility of the model, the shaft speeds of the compressor and turbine are specified separately. The model allows the use of a single shaft by setting the turbine shaft speed to a negative value; this indicates to the model that the two speeds are linked. It

should be noted that the shaft speed of the recompressor is not specified but is rather calculated according to the desired recompression fraction (ϕ_{rc}). Because the performance of the modeled recompression cycle is equivalent to the simple recuperated Brayton cycle (shown in Fig. 1.1) when the recompression fraction goes to zero, this model is capable of simulating both cycle configurations.

The compressor inlet temperature is used as an input because the temperature to which the carbon dioxide can be cooled before entering the compressor is highly dependent on ambient conditions and the design of the precooler. In the limit of a perfect precooler, the lowest possible temperature for a dry-only air-cooled cycle is the ambient dry bulb temperature. The lowest possible temperature for a water-cooled cycle would be the ambient wet bulb temperature. It is expected that the cycle heat rejection control strategy will target a known compressor inlet temperature (most likely as low as possible) in order to maximize efficiency. Directly specifying the compressor inlet temperature recognizes that there is a cooling system in place that is operated in order to minimize the temperature of the carbon dioxide at the compressor inlet. Decoupling the operation of the cycle from the performance of the precooler heat exchanger reduces computational overhead.

Depending on the cooling control strategy that is implemented, it is expected that the compressor inlet temperature will track ambient conditions. In other words, a warmer day will result in higher low-side cycle operating temperatures. While the thermal performance of the precooler is not considered when determining the operating point, once the system model has converged a precooler model may be used to determine the necessary cooling conditions required to achieve the target temperature. If those conditions are not possible (e.g., the compressor inlet temperature is specified as 33°C but the lowest possible temperature achievable with the precooler is 36°C), then the compressor inlet temperature can be adjusted. In this way the precooler size and design can still be considered when evaluating cycle performance.

The compressor inlet pressure is specified as an input to the model because it has been determined that actively controlling the pressure of the CO_2 at the compressor inlet is advantageous in regards

to maximizing cycle efficiency (these results are discussed in Chapter 6). Specifically, increasing the compressor inlet pressure can reduce the efficiency degradation that would otherwise occur when operating a cycle under warmer off-design low side temperatures. This type of control is also referred to as "inventory control" and is one of a few strategies being considered for recuperated SCO_2 Brayton cycles (Carstens, 2007; Dostal, 2004). While a disadvantage of inventory control is that it is slow (i.e., it has a relatively long time constant) compared to shaft speed or bypass control (Moisseytsev & Sienicki, 2009), this is not a concern for adjusting the operating point of the cycle based on slow changes in ambient conditions.

Similar to the Carnot cycle, increasing the CO_2 temperature at the turbine inlet of an SCO_2 Brayton cycle increases the thermal efficiency. A desirable control strategy is to attempt to run the cycle at the highest possible temperature, the limit of which is determined by material properties and available solar resource. In a traditional gas-fired air Brayton cycle, the material properties of the turbine blades limit the high side gas temperature. For CSP applications, the thermal limit of the high side is likely the breakdown of molten salt, as well as the thermal limits of the materials used to construct the receiver or primary heat exchanger. As is the case for the compressor, specifying the turbine inlet temperature assumes that the primary heat exchanger is adequately designed and controlled; this assumption reduces computational overhead.

In order to determine the operating point of the cycle (illustrated by the intersection of the compressor and turbine curves in Fig. 3.14), the model iteratively determines the corresponding mass flow rate through the turbomachinery. A flowchart of the iteration logic used in the off-design cycle model is shown in Figure 3.16.

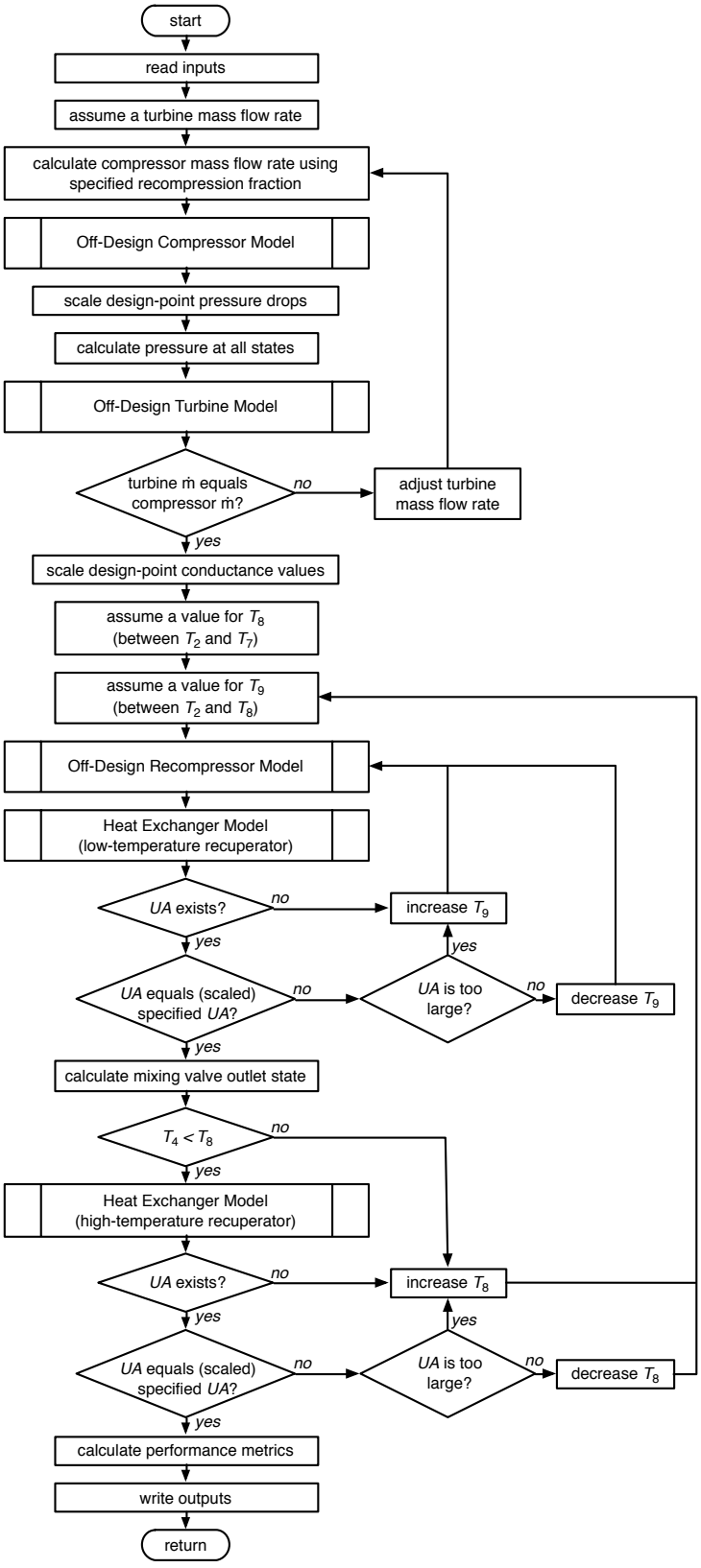


Figure 3.16. Iteration process for the off-design cycle model.

The off-design iteration strategy is very similar to the design-point iteration strategy described in Section 2.3; the only major difference is in the calculation of mass flow rate. In the design-point analysis, the target power output is used to determine the required mass flow rate through the cycle, while in the off-design analysis the mass flow rate is determined by matching the head-flow curve of the main compressor with the flow characteristic of the turbine. This matching process corresponds to the first iteration loop shown in Figure 3.16. The general strategy is to assume a value for the mass flow rate through the turbine, which also fixes the mass flow rate through the main compressor for the specified (as an input) recompression fraction. Given the mass flow rate through the compressor, its shaft speed, and the temperature and pressure at the main compressor inlet (state 1), the main compressor model predicts the outlet temperature and pressure (state 2). Once the pressures at state 1 and 2 are known, all other pressures are calculated by taking into effect the scaled pressure drops through the various heat exchangers. The turbine model uses the temperature and pressure at state 6 as well as the pressure at state 7 to predict an allowable mass flow rate; if the allowable mass flow rate does not match the current guess, a new value is chosen using a combination of the bisection and secant root-finding methods (Chapra & Canale, 2009) and the process repeats. The allowable mass flow rate is the predicted mass flow rate through the turbine as a function of the pressure drop across it.

Once the mass flow rate through the cycle has converged, control passes to the nested iteration loop that determines the temperatures at the remaining unknown state points. This process is described in Section 2.3 and remains fundamentally unchanged, with the exception of the recompressing compressor. Specifically, the efficiency of the recompressing compressor is determined using the known mass flow rate (based on the set recompression fraction ϕ_{rc}) and required pressure rise from states 9 to 10; the required shaft speed is also determined in this manner.

4. Modeling Framework

The design-point and off-design modeling methodologies described in Chapters 2 and 3 are implemented as a modeling framework in the Fortran language. The source code for the framework, which is organized into seven modules, is available online[2] and is also included in Appendices I-VII of this document. A Fortran module is a self-contained unit of code, the use of which is considered best practice with modern Fortran because it simplifies the code structure and enforces compile-time argument checking (Brainerd, 2009). The major advantage of the module-based approach used in the current framework is the ability to easily implement various fluid property libraries and component models at compile time. Specifically, a user may replace any of the non-required modules in the code with an alternative containing custom code. A brief description of the seven modules is provided in Table 4.1 and each will be discussed in turn, though the source code itself is well-commented and is intended to be the primary reference for the framework. More than one filename is listed for the the `compressors`, `turbines`, and `CO2_properties` modules; each filename is an alternative implementation of that module that is currently available. For example, the `radial_turbine.f90` file implements the low-reaction radial turbine model discussed in Chapter 3, while the `snl_radial_turbine.f90` file implements the modified radial turbine model that more closely represents the turbine being investigated by SNL for SCO_2 applications.

2. http://sel.me.wisc.edu/software.shtml (The files available online are likely more up-to-date than the source code listed in the appendices of this dissertation.)

Table 4.1. Fortran modules used by the developed modeling framework.

Module Name	Filename	Description
`core`	`core.f90` (required)	Defines a number of user-defined types and contains a number of subroutines and functions required by the `design_point` and `off_design_point` modules.
`design_point`	`design_point.f90` (required)	Contains the system-level subroutines used to model cycles at the design point.
`off_design_point`	`off_design_point.f90` (required)	Contains the system-level subroutines used to model cycles under off-design or part-load conditions.
`heat_exchangers`	`scaling_hxr.f90` (may be replaced)	Defines the functions responsible for scaling conductance and pressure drop under off-design mass flow rates.
`compressors`	`snl_compressor.f90` `snl_compressor_tsr.f90` (may be replaced)	Contains compressor and recompressor sizing and performance subroutines based on the SNL compressor.
`turbines`	`radial_turbine.f90` `snl_radial_turbine.f90` (may be replaced)	Contains the turbine sizing and performance subroutines based on a radial turbine.
`CO2_properties`	`module_CO2_properties.f90` `CO2_RP_module.f90` (may be replaced)	Contains the required fluid property subroutines for carbon dioxide.

4.1. `core` Module

The core module defines a number of user-defined types that are required by the other modules. User-defined types in Fortran are similar to structures in C, and are used for representing various components because they greatly simplify the calling procedures of the various subroutines.

The user-defined types that are defined in `core` are:

- `Compressor` - holds information related to the main and recompressing compressors
- `Turbine` - holds information related to the turbine
- `HeatExchanger` - holds information related to the recuperators, precooler, and primary heat exchanger
- `RecompCycle` - holds cycle information, including references to the component types above
- `ErrorTrace` - holds information related to the type and location of any encountered errors

The core module also includes two subroutines used by the design-point model to calculate the performance of turbomachinery based on isentropic or polytropic efficiency. The sub-heat exchanger model used by both the design-point and off-design models is also contained in this module.

4.2. `design_point` Module

The design-point cycle model is implemented as the subroutine `design` in the `design_point` module with the following form:

```
subroutine design(W_dot_net, T_mc_in, T_t_in, P_mc_in, P_mc_out, DP_LT, DP_HT, DP_PC,
                  DP_PHX, UA_LT, UA_HT, recomp_frac, N_t, eta_mc, eta_rc, eta_t, N_sub_hxrs,
                  tol, error_trace, recomp_cycle)
```

The required input arguments for the `design` subroutine are listed in Table 4.2; note that `error_trace` and `recomp_cycle` are output variables of type `ErrorTrace` and `RecompCycle`, respectively.

Table 4.2. Description of input arguments used by the `design` subroutine.

Variable	Units	Description
`W_dot_net`	kW	power output of the cycle, defined as turbine power less compressor(s) power
`T_mc_in`	K	main compressor inlet temperature (low-side temperature)
`T_t_in`	K	turbine inlet temperature (high-side temperature)
`P_mc_in`	kPa	main compressor inlet pressure (low-side pressure)
`P_mc_out`	kPa	main compressor outlet pressure (high-side pressure)
`DP_LT`	kPa / %	pressure drop in low-temperature recuperator; negative value indicates relative
`DP_HT`	kPa / %	pressure drop in high-temperature recuperator; negative value indicates relative
`DP_PC`	kPa / %	pressure drop in precooler; negative value indicates a relative pressure drop
`DP_PHX`	kPa / %	pressure drop in primary heat exchanger; negative value indicates relative
`UA_LT`	kW/K	low-temperature recuperator conductance (*UA*)
`UA_HT`	kW/K	high-temperature recuperator conductance (*UA*)
`recomp_frac`	-	fraction of mass flow rate that bypasses the precooler and main compressor

eta_mc	-	main compressor efficiency; positive is isentropic, negative is polytropic
eta_rc	-	recompressor efficiency; positive is isentropic, negative is polytropic
eta_t	-	turbine efficiency; positive is isentropic, negative is polytropic
N_sub_hxrs	-	number of sub-heat exchangers used to discretize recuperators
tol	-	relative convergence tolerance for iteration loops

A number of inputs listed in Table 4.2 will be set based on the assumed characteristics of the cycle (e.g., turbomachinery efficiency or desired power output), but there are some whose values are not immediately obvious. Namely, the optimal values of compressor inlet and outlet pressure, the recompression fraction, and the distribution of the total available conductance between the two recuperators must be determined numerically. To facilitate the optimization of thermal efficiency by varying these values, another subroutine is implemented in this module that allows an arbitrary set of these four inputs to be used as free variables in an optimization routine:

```
subroutine optimal_design(W_dot_net, T_mc_in, T_t_in, DP_LT, DP_HT, DP_PC, DP_PHX,
                          UA_rec_total, eta_mc, eta_rc, eta_t, N_sub_hxrs, P_high_limit,
                          P_mc_out_guess, fixed_P_mc_out, PR_mc_guess, fixed_PR_mc,
                          recomp_frac_guess, fixed_recomp_frac, LT_frac_guess,
                          fixed_LT_frac, N_t, tol, opt_tol, error_trace, recomp_cycle)
```

The inputs for the `optimal_design` subroutine that are not shared by the `design` subroutine are listed in Table 4.3.

Table 4.3. Description of input arguments used by the `optimal_design` subroutine.

Variable	Units	Description
P_high_limit	kPa	maximum allowable high-side pressure
P_mc_out_guess	kPa	initial value for main compressor outlet pressure
fixed_P_mc_out	T/F	whether `P_mc_out_guess` is fixed or available for optimization
PR_mc_guess	-	initial value for main compressor pressure ratio
fixed_PR_mc	T/F	whether `PR_mc_guess` is fixed or available for optimization
recomp_frac_guess	-	initial value for fraction of mass flow rate that bypasses precooler
fixed_recomp_frac_guess	T/F	whether `recomp_frac_guess` is fixed or available for optimization
LT_frac_guess	-	initial value for fraction of total conductance allocated to the low-temperature recuperator
fixed_LT_frac	T/F	Whether `LT_frac_guess` is fixed or available for optimization
opt_tol	-	tolerance for optimization convergence

If all four of the optimization variables are "fixed" (i.e., the values for `fixed_P_mc_out`, `fixed_PR_mc`, `fixed_recomp_frac_guess`, and `fixed_LT_frac` are all set to `.true.`), the subroutine simply runs the design-point model with the specified values. If any of the optimization variables are allowed to vary, the subroutine uses the subplex algorithm, which is is a hill-climbing algorithm based on an extension of the Nelder-Mead optimization method (Rowan, 1990), as made available in the Netlib repository (Browne et al., 1995).

In order to reliably find the global optimum for a given design point, it is often necessary to start the optimization search with a number of different initial conditions. This process is automated in the `auto_optimal_design` subroutine, which is the recommended design-point subroutine to use for determining a single optimal design point:

```
subroutine auto_optimal_design(W_dot_net, T_mc_in, T_t_in, DP_LT, DP_HT, DP_PC, DP_PHX,
                               UA_rec_total, eta_mc, eta_rc, eta_t, N_sub_hxrs,
                               P_high_limit, N_t, tol, opt_tol, return_error_code, cycle)
```

Besides initializing the optimization algorithm with a number of different initial conditions, the `auto_optimal_design` subroutine also implements an outer, one-dimensional optimization loop that varies the high-side pressure of the cycle. This loop uses Brent's Method, which is a combination of the golden section search and successive parabolic interpolation methods (Brent, 1973), as implemented in the `fmin` function of the Netlib repository. This subroutine is more likely to determine the global maximum of efficiency, but it is 20-30 times slower than the `optimal_design` subroutine due to the outer optimization loop and multiple initial conditions that are checked.

4.3. `off_design_point` Module

The iteration strategy shown in Fig. 3.16 is implemented in the subroutine `off_design`, which is a part of the `off_design_point` module and has the form:

```
subroutine off_design(recomp_cycle, T_mc_in, T_t_in, P_mc_in, recomp_frac, N_mc, N_t,
                      N_sub_hxrs, tol, error_trace)
```

The `recomp_cycle` argument is an instance of a `RecompCycle` variable, which contains the necessary design-point information (e.g., compressor rotor diameter, design-point shaft speed, etc.) and will be updat-

ed based on the remaining input arguments, which are listed in Table 4.4. Note that the recompressor shaft speed is not listed as an input; rather, the shaft speed required to attain the specified recompression fraction is calculated. This approach assumes that the recompressor is driven independently by a motor that can be adequately controlled.

Table 4.4. Input arguments to the off-design cycle model.

Variable	Units	Description
T_mc_in	K	main compressor inlet temperature (low-side temperature)
T_t_in	K	turbine inlet temperature (high-side temperature)
P_mc_in	kPa	main compressor inlet pressure (low-side pressure)
recomp_frac	-	fraction of mass flow rate that bypasses the precooler and main compressor
N_mc	rpm	main compressor shaft speed
N_t	rpm	turbine shaft speed; setting value to -1 links turbine and compressor shafts
N_sub_hxrs	-	number of sub-heat exchangers used to discretize recuperators
tol	-	relative convergence tolerance for iteration loops

The `off_design` subroutine is capable of predicting thermal efficiency and power output for a given set of inputs, but it is often desired to maximize efficiency or power output given constraints on the low-side and/or high-side temperature. The `optimal_off_design` subroutine facilitates this optimization using an approach nearly identical to the `optimal_design` subroutine described above. The calling structure for the `optimal_off_design` subroutine is:

```
subroutine optimal_off_design( recomp_cycle, T_mc_in, T_t_in, value_code, N_sub_hxrs,
                               P_mc_in_guess, fixed_P_mc_in, recomp_frac_guess,
                               fixed_recomp_frac, N_mc_guess, fixed_N_mc, N_t_guess,
                               fixed_N_t, tol, opt_tol, error_trace)
```

The inputs for the `optimal_off_design` subroutine that are not shared by the `off_design` subroutine are listed in Table 4.5.

Table 4.5. Description of input arguments used by the optimal_off_design subroutine.

Variable	Units	Description
value_code	-	whether to maximize efficiency (1) or power output (2)
P_mc_in_guess	kPa	initial value for main compressor inlet pressure (low-side pressure)
fixed_P_mc_in	T/F	whether P_mc_out_guess is fixed or available for optimization
recomp_frac_guess	-	initial value for fraction of mass flow rate that bypasses precooler
fixed_recomp_frac_guess	T/F	whether recomp_frac_guess is fixed or available for optimization
N_mc_guess	rpm	initial value for main compressor shaft speed
fixed_N_mc	T/F	whether N_mc_guess is fixed or available for optimization
N_t_guess	rpm	initial value for turbine shaft speed; (-1 links to compressor shaft)
fixed_N_t	T/F	whether N_t_guess is fixed or available for optimization
opt_tol	-	tolerance for optimization convergence

As is the case for the design-point subroutines described in the previous section, the optimal_off_design subroutine allows control variables (recomp_frac, N_mc, and N_t, in this case) to be fixed or to be varied by the subplex method in order to maximize efficiency or power output.

The inputs listed in Table 4.5 do not place any type of constraint on the power produced by the cycle. However, the highest thermal efficiency that is possible at a given power output is often of interest. In other words, the model should be capable of varying the inputs listed in Table 4.4 to provide a target power output at the maximum possible efficiency. This capability is accomplished using the iteration strategy depicted in Figure 4.1.

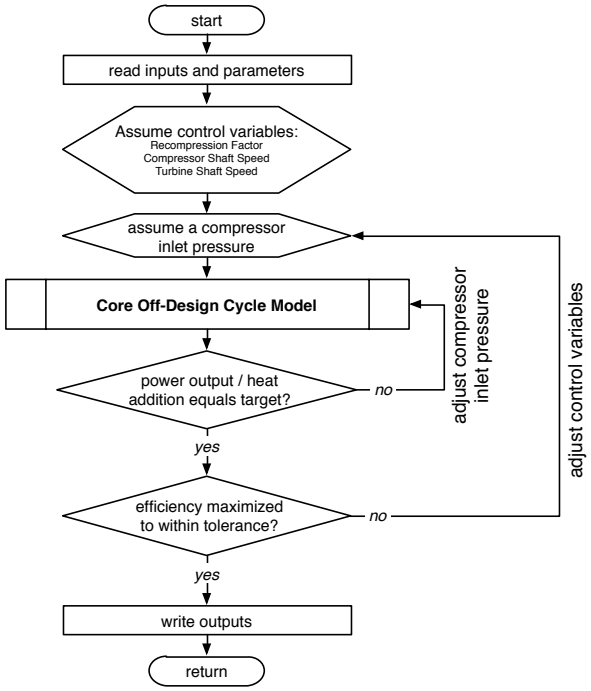


Figure 4.1. Iteration strategy for maximizing off-design cycle efficiency given a target power output or rate of heat addition.

In general, the low-side pressure is varied to achieve a target power output (or rate of heat addition) while the recompression fraction, compressor shaft speed, and turbine shaft speed are varied to maximize efficiency. The subplex algorithm is used for the optimization of the control variables, and the low-side pressure is iterated upon using a combination of the bisection and secant methods. The main compressor and turbine shaft speeds can be varied independently (i.e., for the split-shaft configuration) or linked (i.e., for the single-shaft configuration). It is assumed that an independent motor always drives the recompressor. This logic is implemented in the subroutine `optimal_target_off_design` that has the form:

```
subroutine optimal_target_off_design(recomp_cycle, T_mc_in, T_t_in, target, target_code,
                                     N_sub_hxrs, lowest_pressure, highest_pressure,
                                     recomp_frac_guess, fixed_recomp_frac, N_mc_guess,
                                     fixed_N_mc, N_t_guess, fixed_N_t, tol, opt_tol,
                                     error_trace)
```

The `lowest_pressure` and `highest_pressure` inputs define the interval that is assumed to contain low-side pressure that corresponds to the target specified by the input `target`. The argument `target_code` is used to indicate whether target corresponds to power output (1) or rate of heat addition (2), as the iteration strate-

gy shown Fig. 4.1 for target power output also applies to determining the low-side pressure that corresponds to a desired rate of heat input at the primary heat exchanger.

4.4. heat_exchangers Module

The heat_exchangers module contains the functions hxr_pressure_drops and hxr_conductance, which calculate the scaled pressure drops and conductance of a heat exchanger, respectively. These functions are based on the scaling equations derived in Section 3.4 and can be replaced with any functions that fit the following forms:

```
function hxr_pressure_drops(hxr, m_dots)
function hxr_conductance(hxr, m_dots)
```

The inputs are hxr, a HeatExchanger type, and m_dots, which is an array of length two containing the cold (1) and hot (2) stream mass flow rates. The function hxr_pressure_drops returns a double precision array of length two (corresponding to the two streams), and the hxr_conductance function returns a single double precision value. Any code that satisfies these requirements can be used to represent the off-design pressure drops and conductance of the heat exchangers in the cycle.

4.5. compressors Module

The compressors module is used to size (i.e., determine the required parameters such as rotor diameter and shaft speed) the compressor and recompressor based on the design point of the cycle. The subroutines responsible for determine the necessary parameters are compressor_sizing and recompressor_sizing, both of which require a RecompCycle variable as an argument. The design-point information contained in this variable is used to size the respective compressors, and the calculated turbomachinery parameters are stored in the same cycle variable. Besides sizing the two compressors, the compressors module is responsible for modeling the off-design performance of the two compressors. The calling structures for these two subroutines are:

```
off_design_compressor(comp, T_in, P_in, m_dot, N, error_trace, T_out, P_out)
off_design_recompressor(comp, T_in, P_in, m_dot, P_out, error_trace, T_out)
```

The argument `comp` is a `Compressor` type and the remaining inputs correspond to the inputs and outputs described in the Component Model Interfaces section of Chapter 3. Note that the off-design compressor model requires shaft speed as an input and calculates an outlet pressure, while the off-design recompressor model uses the desired outlet pressure to calculate the corresponding shaft speed. This module may be replaced with an alternative implementation that provides these four subroutines, allowing any type of compressor or recompressor to be represented in the off-design cycle model.

4.6. `turbines` Module

Similar to the `compressors` module, the `turbines` module is responsible for sizing the turbine and modeling its off-design behavior. As discussed in Section 3.1, the off-design performance subroutine requires the inlet temperature and pressure, the outlet pressure, and the shaft speed as inputs and provides the mass flow rate and outlet temperature as outputs. The subroutine is:

```
off_design_turbine(turb, T_in, P_in, P_out, N, error_trace, m_dot, T_out)
```

where `turb` is a `Turbine` type. The sizing subroutine is named `turbine_sizing` and, similar to the sizing subroutines in the `compressors` module, requires a `RecompCycle` variable as an input that contains design-point information.

4.7. `CO2_properties` Module

The `CO2_properties` module is responsible for providing the properties of carbon dioxide as a function of various independent properties. This module can be replaced in order to use a fluid properties library besides FIT, a limited version of which is included in the framework. For example, an alternative module (`CO2_RP_module.f90`) that interfaces with the REFPROP library is included with the source code online and listed in Appendix VII. Note that the source code for REFPROP is not included and must be obtained separately in order to use this alternative module. The following subroutines are provided:

```
CO2_TD(T, D, error_code, temp, pres, dens, enth, entr, ssnd)
CO2_TP(T, P, error_code, temp, pres, dens, enth, entr, ssnd)
CO2_PH(P, H, error_code, temp, pres, dens, enth, entr, ssnd)
CO2_PS(P, S, error_code, temp, pres, dens, enth, entr, ssnd)
CO2_HS(H, S, error_code, temp, pres, dens, enth, entr, ssnd)
```

5. Design-Point Analysis

5.1. Comparison to Literature

Three different design conditions are considered to compare calculated results for the design-point cycle model with those appearing in previously published literature. Two of the cycle designs correspond to those suggested by Dostal (2004) as the “Basic” and “High Performance” cycles. The key distinction between the Basic and High Performance cycles is the turbine inlet temperature, which is 550°C and 700°C, respectively. The third design, "Dry Cooled", is identical to the High Performance design except the compressor inlet temperature is increased to reflect conditions expected when heat is rejected from the cycle by dry cooling. For this analysis the compressor inlet temperature for the Dry Cooled design is 55°C (131°F), which is above the maximum dry-bulb temperature for most locations. The 32°C (89.6°F) compressor inlet temperature for the Basic and High Performance designs was chosen by Dostal because it is near the critical point of carbon dioxide.

The relevant parameters for each of the cycle designs are summarized in Table 5.1. For this comparison, the net mechanical power output (i.e., the power generated by the turbine less the power required by the compressor and recompressor) of the cycles are fixed at 300 MW, while the designs presented by Dostal assume a constant thermal input of 600 MW.

Table 5.1. Design-point conditions for model comparison with literature.

	Basic	**High Performance**	**Dry Cooled**
Compressor Inlet Temperature	32°C	32°C	55°C
Turbine Inlet Temperature	550°C	700°C	700°C
Compressor Outlet Pressure	20 MPa		
Pressure Ratio	2.6		
Compressor Isentropic Efficiency	0.89		
Recompressor Isentropic Efficiency	0.89		
Turbine Isentropic Efficiency	0.90		
Net Mechanical Power Output	300 MW		

The pressure ratio of 2.6 was suggested by Dostal as the optimal pressure ratio for the Basic and High Performance cases and corresponds to a low-side (i.e., main compressor inlet) pressure of 7.69 MPa for the fixed 20 MPa high side pressure. It is expected (based on results discussed in Section 5.2) that the optimal pressure ratio for the Dry Cooled case will be lower; but for consistency, the same low-side pressure of 7.69 MPa is used for all the results shown in Figures 5.1-5.3. Figure 5.1 shows the thermal efficiency of the cycle as a function of recompression fraction for various values of the total recuperator conductance. The optimal distribution of conductance between the two recuperators is varied in order to maximize efficiency at each design point. The dashed line in Fig. 5.1 represents the recompression fraction, ϕ, that maximizes cycle efficiency for a range of recuperator conductance values. In order to verify the FIT properties library, the thermal efficiencies calculated using carbon dioxide properties provided by REFPROP are plotted as open circles. The agreement between the two fluid property libraries is typically on the order of the six significant figures.

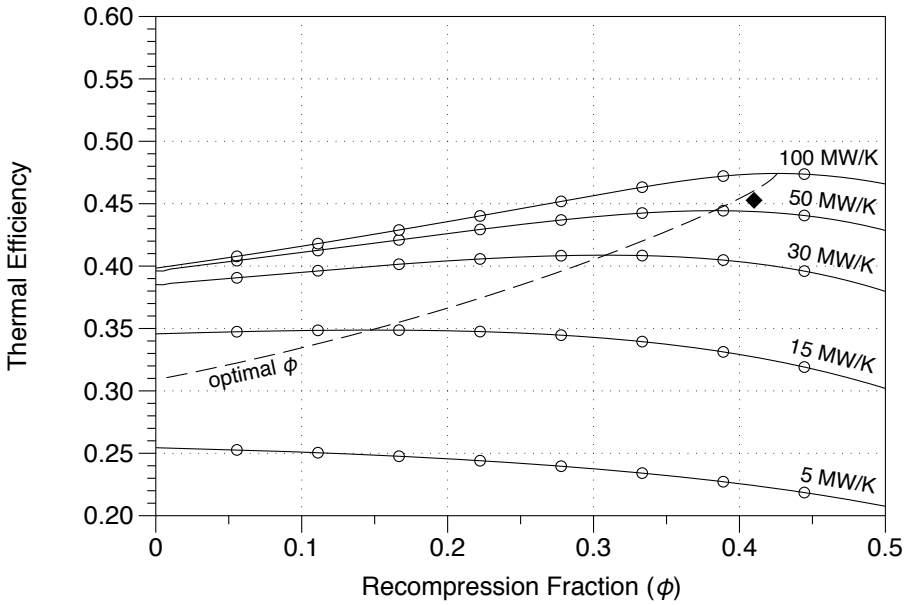


Figure 5.1. Thermal efficiency as a function of recompression fraction for the "Basic" design with fixed low-side and high-side pressures of 7.69 MPa and 20 MPa, respectively.

The black diamond in Fig. 5.1 indicates the optimal recompression fraction and thermal efficiency as reported by Dostal. It should be noted that Dostal used a detailed heat exchanger model for a partic-

ular geometry in his analysis that included the effect of pressure drops. Despite these differences, Dostal's predicted optimal recompression fraction and efficiency shows good agreement with the predicted relationship between optimal recompression fraction and efficiency in the present analysis. Figure 5.1 also suggests that the recuperators Dostal used in his analysis had an equivalent total recuperator conductance of approximately 60 MW/K for the Basic design.

Figure 5.2 is a similar plot for the High Performance design, again showing good agreement with the optimal recompression fraction suggested by Dostal.

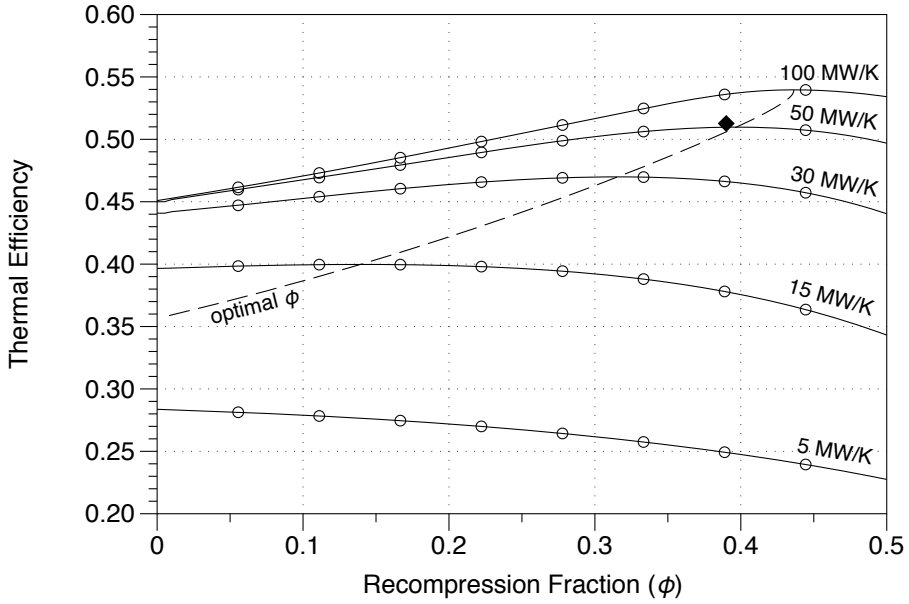


Figure 5.2. Thermal efficiency as a function of recompression fraction for the "High Performance" design with fixed low-side and high-side pressures of 7.69 MPa and 20 MPa, respectively.

As expected based on the Carnot efficiency, increasing the turbine inlet temperature of the cycle increases its efficiency, but otherwise the relationship between recompression fraction, recuperator conductance, and efficiency shown in Fig. 5.2 is similar to the relationship shown in Fig. 5.1. It is worth noting that for both designs there is a recuperator conductance below which the optimal efficiency corresponds to a recompression fraction of zero, indicating a simple cycle is preferred over a recompression

cycle. The observation that small recuperators drive the optimal recompression fraction to zero was also noted by Bryant et al. (2011).

The plot in Figure 5.3 corresponds to the Dry Cooled design for a fixed low-side pressure of 7.69 MPa; a higher thermal efficiency is likely possible by optimizing the low-side pressure for these conditions, as illustrated by the results shown later in Figure 5.6.

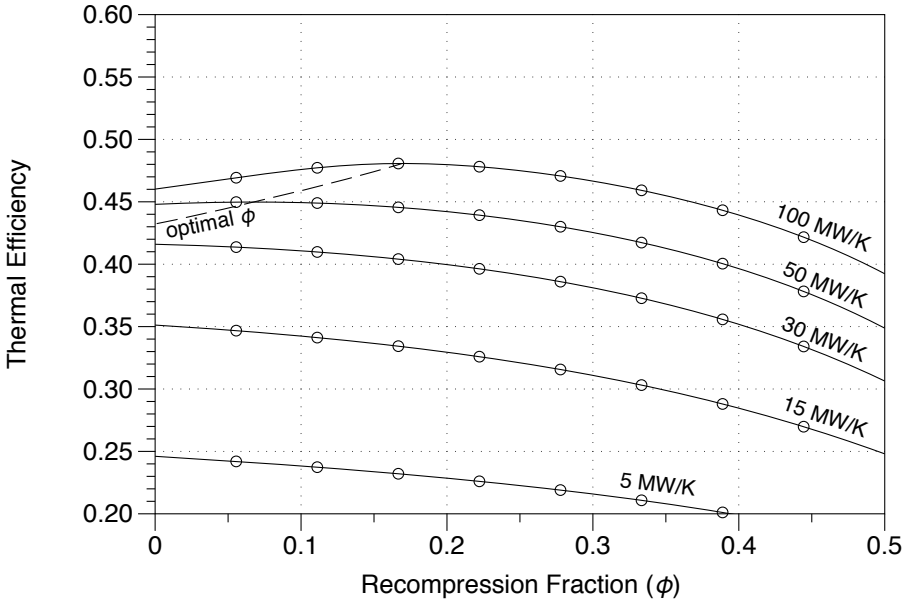


Figure 5.3. Thermal efficiency as a function of recompression fraction for the "Dry Cooled" design with fixed low-side and high-side pressures of 7.69 MPa and 20 MPa, respectively.

Figure 5.3 shows that the optimum cycle efficiency for the warmer low-side temperature of a dry-cooled cycle configuration tends toward a lower optimal recompression fraction. This trend is consistent with the results reported by Sarkar and Bhattacharyya (2009), despite their use of an effectiveness-based heat exchanger analysis that assumes constant properties. The results shown in Figures 5.1-5.3 are not fully optimized because both the compressor outlet pressure and pressure ratio (and hence compressor inlet pressure) are constrained to the values shown in Table 5.1. The effect of these constraints is considered in the following section.

5.2. Effect of High-Side Pressure Limit

The highest CO_2 pressure in the cycle occurs at the main compressor outlet (state 2 in Fig. 2.1). Increasing this pressure typically leads to an increase in thermal efficiency, but higher pressures also require components able to withstand greater stress (typically larger and more expensive). Most previously published analyses have focused on limiting high-side pressures to the 20-30 MPa range, with a typical baseline value of 20 MPa providing a balance between thermal efficiency and economic feasibility (Dostal, 2004). However, it is important to understand the impact that the high-side pressure limit has on the thermodynamic performance of the cycle because future technological improvements may allow for more cost-effective high-pressure components. To this end, the maximum possible thermal efficiencies of two design-point heat rejection conditions are investigated for a number of high-side pressure limits. The two cooling conditions of interest are wet-cooled, represented by a compressor inlet temperature of 32°C, and dry-cooled, represented by an inlet temperature of 55°C. Common to each condition is a turbine inlet temperature of 700°C and the assumption that there are no pressure drops in the cycle piping or heat exchangers.

At the design point, both power output and the effect of conductance scale linearly with mass flow rate through the cycle, which allows the results presented in this section to be generalized using a normalized recuperator conductance. The normalized conductance is defined as the total recuperator conductance divided by the power output. For example, a 10 MW cycle with 2,000 kW/K of total conductance in its recuperators has a normalized conductance of 0.2 (kW/K)/kW. For each case, the normalized recuperator conductance is varied as a parameter from 0.001 to 1.0 (kW/K)/kW and two turbomachinery polytropic efficiencies are considered: 0.8 and 0.9. At every point, the optimal compressor inlet and outlet pressure (up to the high-side limit) are determined, as well as the optimal recompression fraction and distribution of total conductance between the two recuperators. The resulting thermal efficiency of the

cycle designed to operate with a compressor inlet temperature of 32°C is shown in Figure 5.4, and the 55°C compressor inlet temperature case is shown in Figure 5.5.

As expected, a wider range of high-side design pressures leads to increases of the cycle thermal efficiency. However, for the 32°C case shown in Fig. 5.4, the thermal performance of the 30, 40, and 50 MPa cases coincide for normalized conductance values above 0.2 (kW/K)/kW. This result indicates that cycles with design high-side pressures above 30 MPa are unnecessary when the recuperator reaches a threshold size. The 55°C case in Fig. 5.5 shows a similar trend, but with the recuperator threshold size increased to approximately 0.5 (kW/K)/kW.

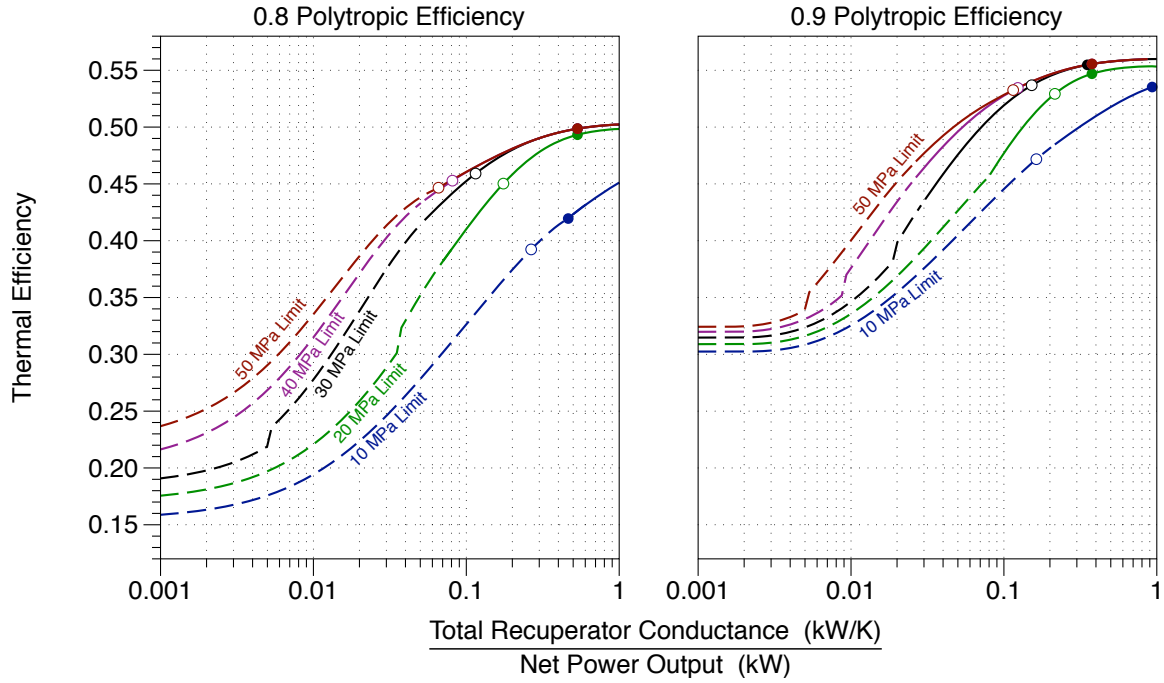


Figure 5.4. Optimal thermal efficiency for a SCO_2 cycle with a compressor inlet temperature of 32°C. The dashed line corresponds to a recompression fraction of zero (i.e., a simple cycle). The open circles mark the normalized conductance corresponding to a minimum temperature difference of 10°C in the recuperators while the filled circles correspond to a 2°C minimum temperature difference.

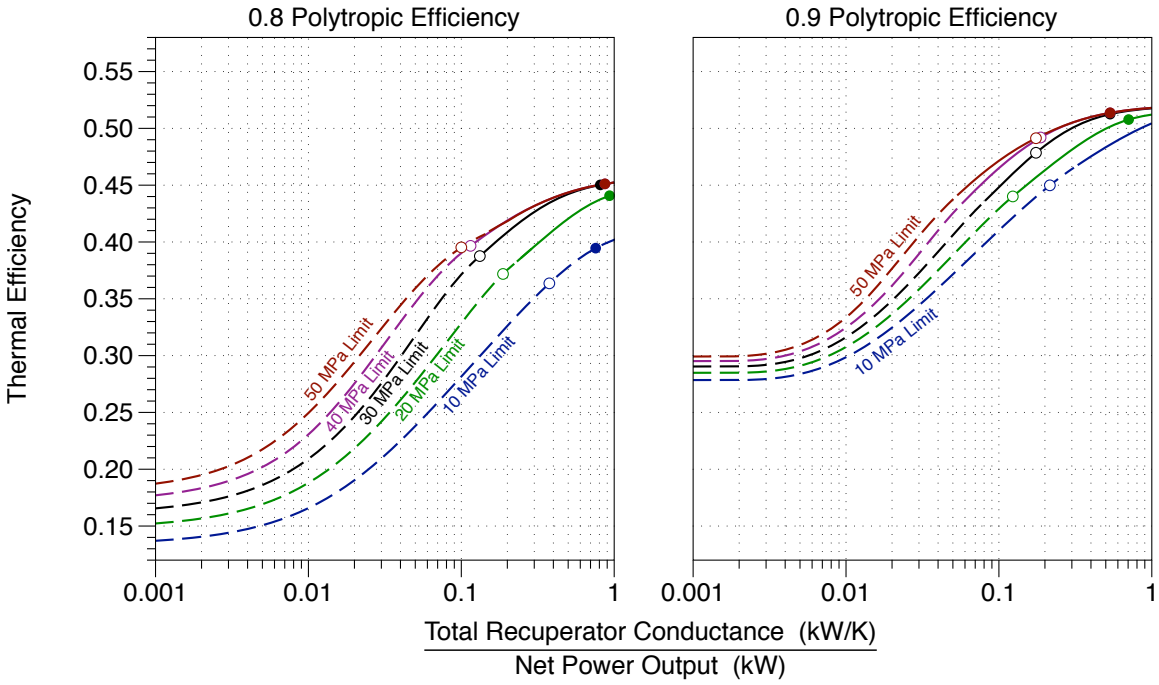


Figure 5.5. Optimal thermal efficiency for a SCO_2 cycle with a compressor inlet temperature of 55°C. The dashed line corresponds to a recompression fraction of zero (i.e., a simple cycle). The open circles mark the normalized conductance corresponding to a minimum temperature difference of 10°C in the recuperators while the filled circles correspond to a 2°C minimum temperature difference.

In Figures 5.4 and 5.5, the dashed lines indicate an optimal recompression fraction of zero, which implies that a simple cycle is preferred over the more complex recompression cycle for those values of normalized recuperator conductance. Because the advantage of the recompression cycle is that it allows for a more balanced recuperator, it makes sense that for cycles with smaller recuperators the additional energy requirements of the recompressing compressor is not outweighed by the increase in the amount of recuperated energy. The open circles in Figures 5.4 and 5.5 mark the normalized conductance corresponding to a minimum temperature difference in the recuperators of 10°C, while the closed circles indicate a minimum temperature difference of 2°C. The circles are included in the figures to provide another reference for the size and performance of the recuperators.

The sharp increases in efficiency visible in Fig. 5.4 correspond to an abrupt shift in the optimal low side pressure of the cycle. This sudden change is apparent in Figure 5.6, which is a plot of the

pressures associated with the efficiencies shown in Figures 5.4 and 5.5 for the 0.9 polytropic efficiency cases. Figure 5.6 also explains why the thermal efficiencies at higher pressure limits and recuperator sizes coincide; as the recuperators becomes sufficiently large, the optimum high-side pressure no longer corresponds to the high-pressure limit.

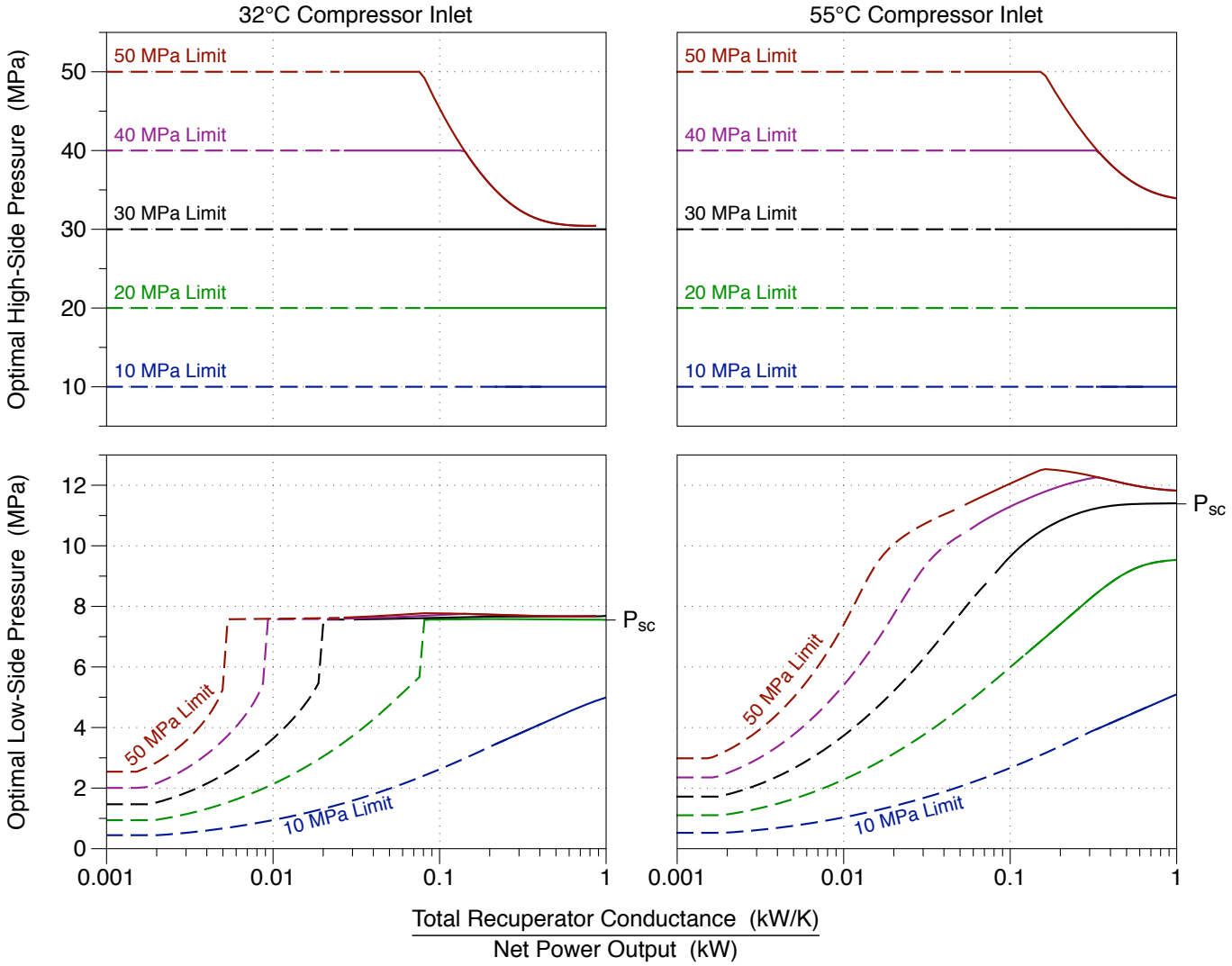


Figure 5.6. Optimal compressor inlet and outlet pressures for the 32°C case (left) and the 55°C case (right). The polytropic efficiency of the turbomachinery is 0.9.

The dashed lines in Fig. 5.6 again indicate an optimal recompression fraction of zero. For the 32°C case, it is interesting to note that the transition from a simple to a recompression cycle occurs independently of the abrupt shift in optimal low-side pressure. This abrupt change is not present for the 55°C

case, as the CO_2 property variations are less severe further from the critical point. However for both cases as recuperator conductance increases, the optimal low-side pressure converges to the pseudocritical pressure (P_{sc}) at the compressor inlet temperature. This relationship between the low-side temperature and optimal low-side pressure explains why the fixed pressure ratio of 2.6 used in Table 5.1 for consistency is not appropriate for a dry-cooled cycle.

5.3. Effect of Low-Side Temperature

The results of the previous section suggest that, with sufficient recuperation, high-side pressures above 30 MPa do not result in appreciable increases in performance and, in some cases, the thermal efficiency of the cycle is reduced. The results also indicate that the compressor inlet temperature has a significant effect on the optimal low-side pressure; specifically, warmer temperatures require higher pressures.

In order to explore the effect of the low-side temperature on cycle performance, two high-side pressures, 20 and 30 MPa, and two normalized recuperator sizes, 0.2 and 0.5 (kW/K)/kW, are considered. The turbomachinery is modeled using isentropic efficiencies of 0.89 for the compressors and 0.93 for the turbine. Isentropic efficiencies are used for this analysis, and the remainder of the analyses in this dissertation, because the pressure ratios for the different designs under consideration do not vary widely. The predicted optimal thermal efficiency for these conditions as a function of compressor inlet temperature is shown in Figure 5.7 for turbine inlet temperatures of 550°C and 700°C. The dot-dash lines correspond to the smaller recuperator case and the solid lines correspond to the larger recuperator case.

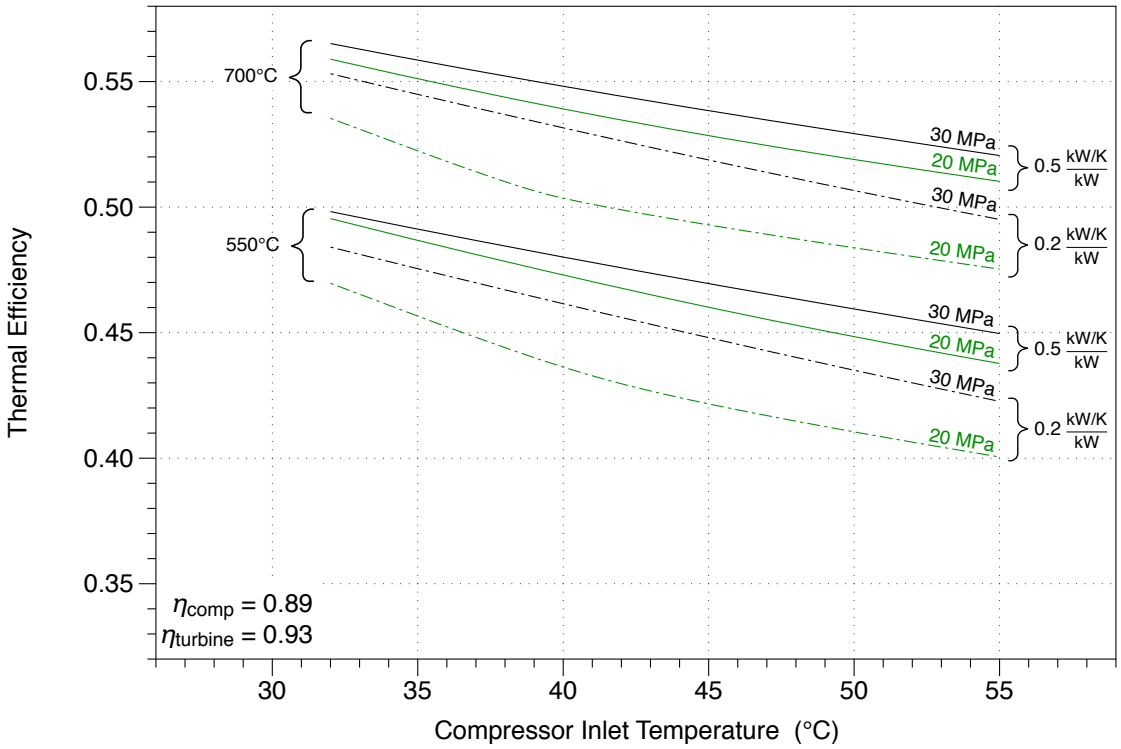


Figure 5.7. Thermal efficiency as a function of compressor inlet temperature for various designs.

It is not surprising that the best cycle performance is obtained with the largest recuperator operating at the lowest compressor inlet temperature and highest turbine inlet temperature. However, the significance of Fig. 5.7 is the relative differences in performance among the cases. The optimal cycle design will ultimately be driven by economics and application-specific design constraints, but Fig. 5.7 provides some insight into the relative tradeoffs to various design decisions. For example, increasing the high-pressure limit from 20 MPa to 30 MPa results in increased efficiency; however, the efficiency increase is not as significant as a cycle using a larger overall recuperator. In cases where capital costs may limit the recuperator size, increasing the high-side pressure limit can effectively compensate for the effect of comparatively lower cycle efficiency with a smaller recuperator. This trend is also apparent in Figures 5.4 and 5.5 for normalized conductance values in the range of 0.01 to 0.2 (kW/K)/kW. Figure 5.7 also shows that thermal efficiencies in excess of 50 percent are possible under a range of compressor inlet temperatures appropriate for dry cooling.

The low-side pressure, recompression fraction, and fraction of recuperator conductance in the low-temperature (LT) recuperator corresponding to the efficiencies in Fig. 5.7 for the 700°C case are plotted in Figure 5.8.

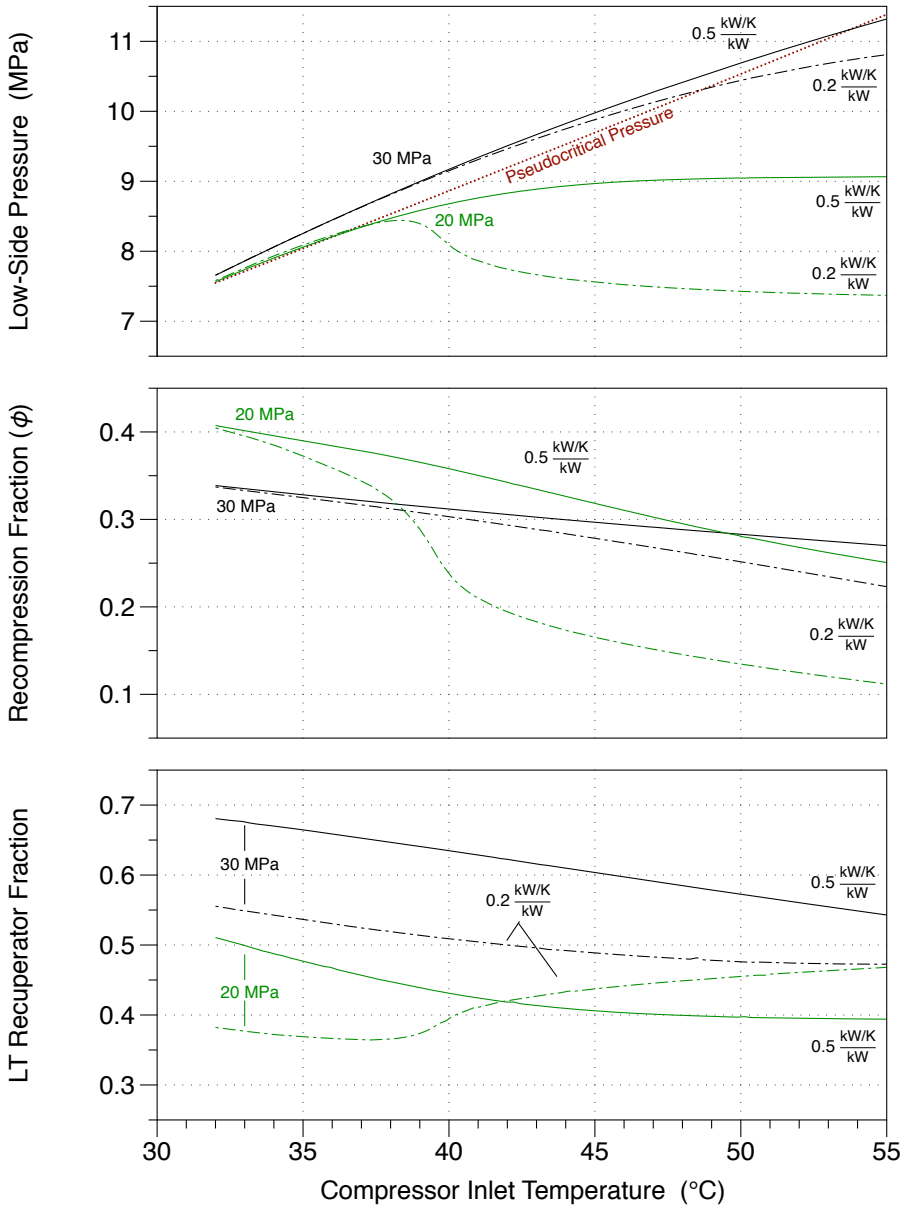


Figure 5.8. Optimal low-side pressure (top), recompression fraction (middle), and low-temperature recuperator conductance fraction (bottom) as a function of low-side temperature with a high-side temperature of 700°C.

The performance of the recompression cycle is strongly dependent on the balance between its pressure ratio, the amount of recuperation taking place, and the potential for the recompressor to improve

that recuperation by balancing the capacitance rates of the hot and cold streams. For a high-side pressure of 30 MPa, the optimal low-side pressure tracks closely with the pseudocritical pressure of carbon dioxide at the compressor inlet temperature. For the 20 MPa case, the optimal low-side pressure begins to deviate from the pseudocritical pressure above low-side temperatures of approximately 38°C (this deviation is also visible for the 55°C case in Fig. 5.6). The reason for this divergence is that the advantage of favorable compression near the pseudocritical region is outweighed by the advantage of operating at a higher pressure ratio. However, for temperatures near the critical point the reduced work required to compress carbon dioxide in the pseudocritical region dominates and the optimal pressure ratio becomes secondary.

The results shown in Figs. 5.4-5.8 assume no pressure drops through heat exchangers or piping in the cycle. Including pressure drop effects will decrease the efficiency of the cycle, but the general trends do not change significantly. To illustrate this point, Figures 5.9 and 5.10 are plots of thermal efficiency as a function of low-side temperature for a number of designs (similar to Fig. 5.7) assuming relative pressure drops through the heat exchangers of one and two percent, respectively. Again, the solid lines represent recuperators with a normalized conductance of 0.5 (kW/K)/kW and the dot-dash lines represent a normalized conductance of 0.2 (kW/K)/kW.

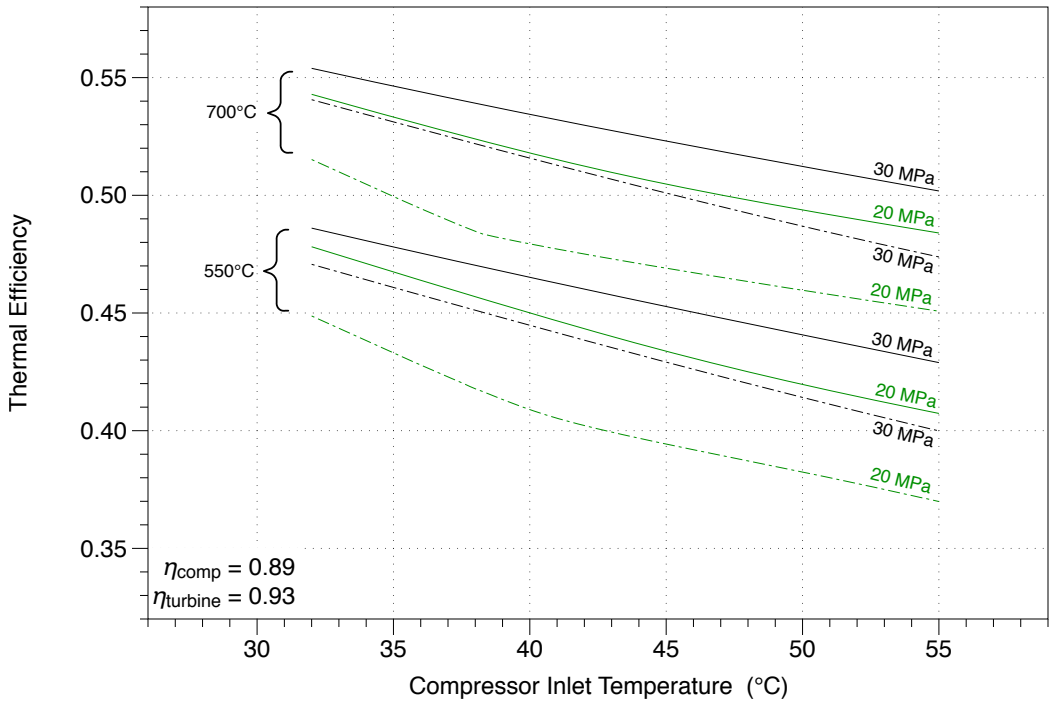


Figure 5.9. Thermal efficiency as a function of compressor inlet temperature for various designs assuming a one percent relative pressure drop through the heat exchangers in the cycle.

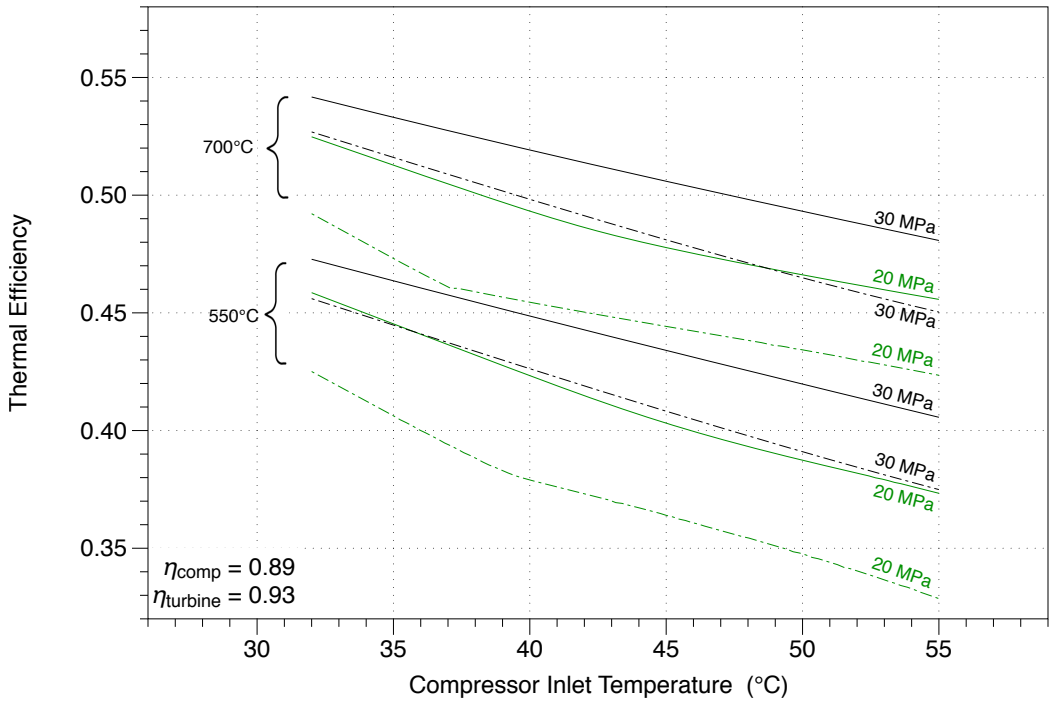


Figure 5.10. Thermal efficiency as a function of compressor inlet temperature for various designs assuming a two percent relative pressure drop through the heat exchangers in the cycle.

The sharp bend in the efficiency curve that occurs around a low-side temperature of 37°C for the 20 MPa, 0.2 (kW/K)/kW design is due to the abrupt change in optimal low-side pressure discussed previously, which is made larger by the pressure drop effects. This exaggeration is clearly shown in Figure 5.11, which is a plot of the low-side pressure, recompression fraction, and fraction of recuperator conductance in the LT recuperator corresponding to the efficiencies for the one percent relative pressure drops case in Fig. 5.9 (with a turbine inlet temperature of 700°C). Specifically, the change in optimal low-side pressure deviates much more sharply for the 20 MPa, 0.2 (kW/K)/kW design than is shown in Fig. 5.8.

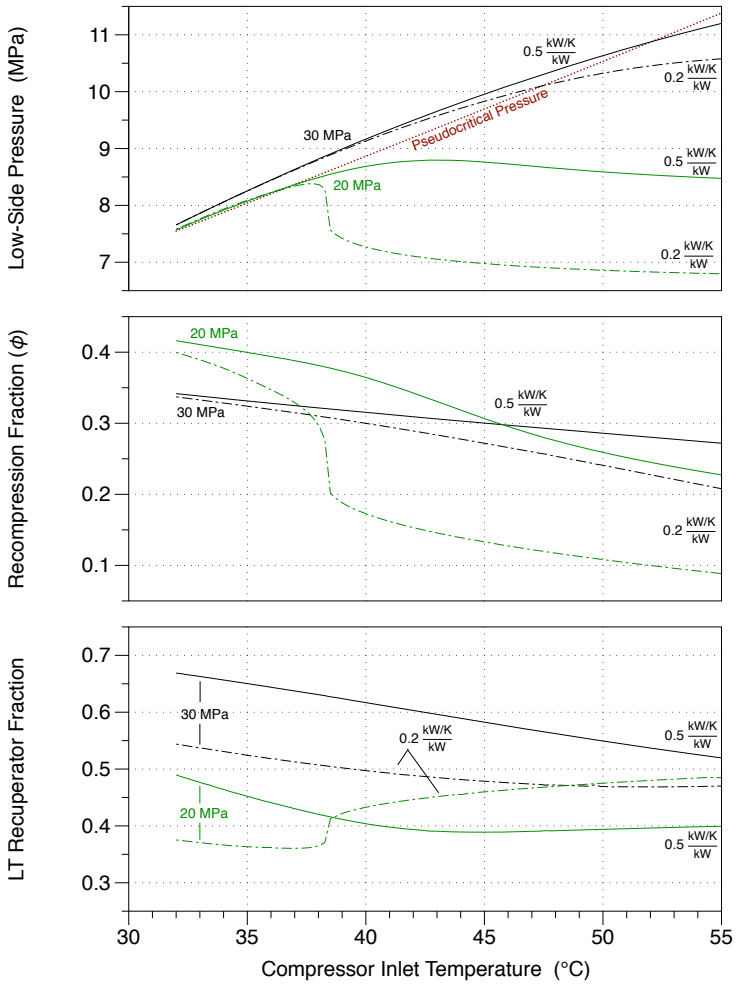


Figure 5.11. Optimal low-side pressure (top), recompression fraction (middle), and low-temperature recuperator conductance fraction (bottom) as a function of low-side temperature with a high-side temperature of 700°C and a one percent relative pressure drop through the heat exchangers in the cycle.

The results of the analyses presented in this chapter indicate that the optimal balance between the pressure ratio and recompression fraction of the cycle is strongly dependent on the design-point low-side temperature, the high-pressure limit, and the size of the recuperators. Therefore, the physical parameters associated with a given design (e.g., compressor rotor diameter and shaft speed) are also strongly dependent on the design-point of the cycle. For example, designing for a warmer low-side temperature in a dry-cooled application with small recuperators will require a smaller recompressor (or none at all). This dependence is the reason the design-point and off-design models are intended to be coupled, as described in Chapter 4. Specifically, the `optimal_design` or `auto_optimal_design` subroutines are used to determine the optimal cycle design for a given set of design-point conditions, after which the corresponding required turbomachinery parameters are determined and used in the off-design models.

5.4. Convergence Verification

The data presented in this chapter were generated using 10 sub-heat exchangers per recuperator and a convergence tolerance of 10^{-5}. In order to verify the models are sufficiently converged with these values, the thermal efficiency and mass flow rate errors for 135 different design points (parametrically selected and representative of the designs considered in this chapter) are plotted as a function of the number of sub-heat exchangers and convergence tolerance in Figure 5.12. The reference values used to determine relative errors are provided from the model using 30 sub-heat exchangers and a convergence tolerance of 10^{-8}. While varying the number of sub-heat exchangers in Fig. 5.12 the convergence tolerance is fixed at 10^{-5}, and while varying the convergence tolerance the number of sub-heat exchangers is set to 10.

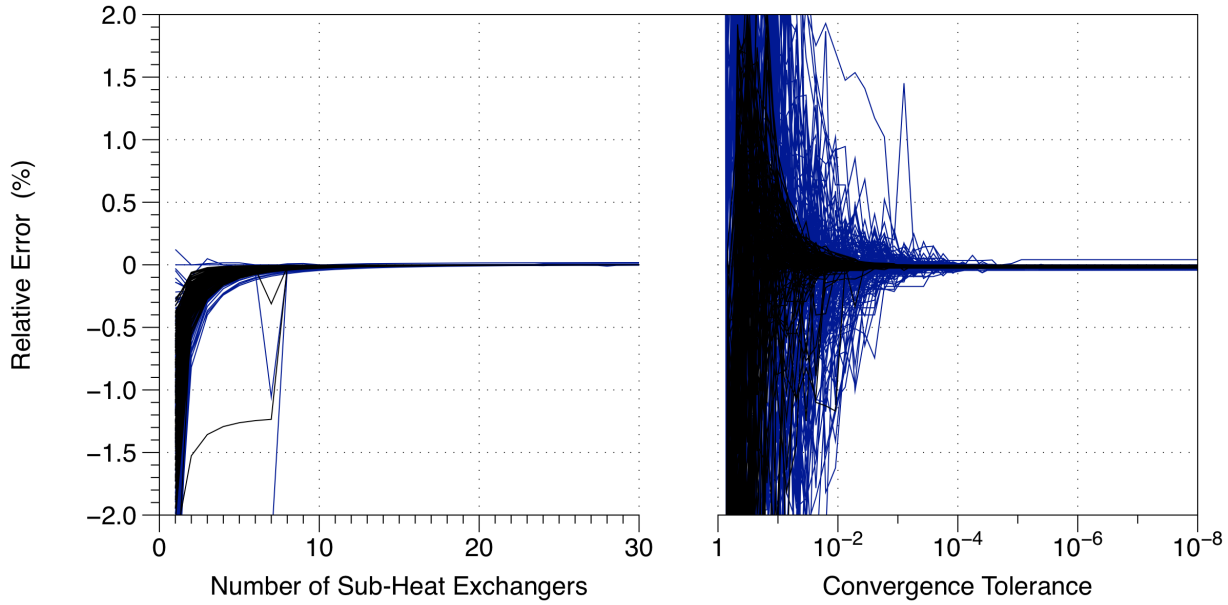


Figure 5.12. Relative error of thermal efficiency (black lines) and mass flow rate (blue lines) for a number of design points. The reference values used for the calculating errors are determined from runs using 30 sub-heat exchangers and a convergence tolerance of 10^{-8}.

Figure 5.12 shows that convergence to within 0.05 percent is achieved for the range of designs using a minimum of 10 sub-heat exchangers and a convergence tolerance on the order of 10^{-4}. The minimum number of sub-heat exchangers that show convergence is used because the runtime of the model increases proportionally with the number of sub-heat exchangers on the order of $O(N^{2/3})$. Figure 5.13 shows the `optimal_design` subroutine runtime as a function of the number of sub-heat exchangers for two designs, one representative of a wet-cooled application and one representative of a dry-cooled application. The plotted data were generated on a 2.6 GHz Intel Core i7 using the gfortran (version 4.9) compiler; the bottom plot uses the FIT properties library and the top plot uses REFPROP.

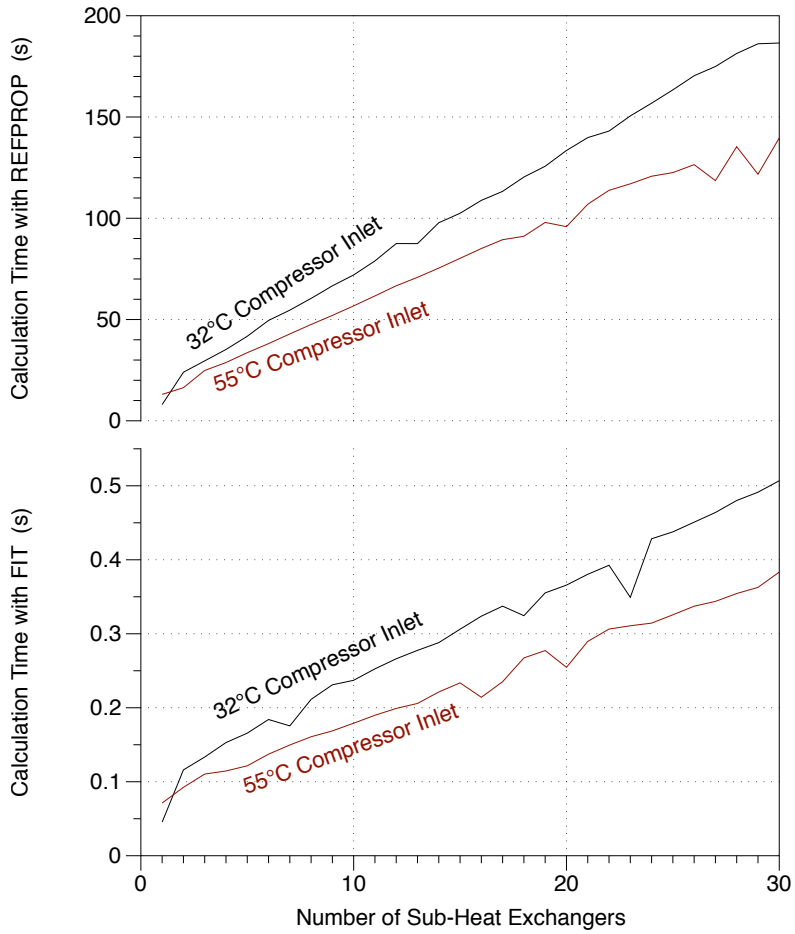


Figure 5.13. Design-point cycle model runtime using FIT (bottom) and REFPROP (top) for two representative designs.

The variations in the calculation time are due to numerical noise. The 32°C compressor inlet case is slower than the 55°C inlet case because of the vicinity to the critical point, where property calls are more computationally expensive. This trend was observed for both REFPROP and FIT.

6. Off-Design Analysis

The results presented in Chapter 5 confirm what is expected based on the Second Law of thermodynamics; namely, designing for a colder low-side temperature (with a fixed high-side temperature) will result in a higher thermal efficiency. It is expected that this relationship between thermal efficiency and compressor inlet temperature is also true for off-design operation. That is, the off-design thermal efficiency of the cycle will decrease as the temperature of the carbon dioxide at the main compressor inlet increases, and conversely the efficiency will increase as the compressor temperature decreases. What is not clear, however, is the relative magnitude of the efficiency degradation of a low-temperature design operating at above design-point temperatures versus the increase in efficiency experienced by a high-temperature design operating at below design-point temperatures. In order to explore this relationship, the off-design performance of two design options are considered. One design assumes a compressor inlet temperature of 32°C and the other design assumes a compressor inlet temperature of 50°C. With respect to the heat rejection system associated with the power cycle, the low-temperature design is likely more appropriate for wet-cooling, while the high-temperature design is more representative of a dry-cooled system. However, the analyses presented in this chapter do not take into account a specific heat rejection mechanism. This approach allows for high-level observations that hold true regardless of the type of heat rejection system utilized by a system.

The off-design model developed for this work effort predicts the off-design performance (i.e., power output and thermal efficiency, among other metrics) of a system as a function of compressor inlet temperature and pressure, turbine inlet temperature, shaft speed, and recompression fraction. However, combinations of these inputs may result in non-physical or unreasonable operating points. As an example of this possibility, Figure 6.1 is a plot of the predicted power output and compressor outlet pressure for two characteristic designs as a function of compressor inlet pressure at various off-design operating temperatures. At the design points, indicated by the black dots in the figure, both cycles operate at a high-

side pressure of 25 MPa and a power output of 10 MW. The specifics of the other design-point parameters are unimportant for this example, as these plots are intended only to illustrate the general off-design trends of the cycles. The primary difference between the two designs is the proximity of the design-point compressor inlet conditions relative to the critical point of carbon dioxide. The lower temperature design takes advantage of favorable compression near the critical point, so at higher pressures the power output decreases. The power output of the higher temperature design will also decrease eventually at higher pressures, but that occurs at compressor inlet pressures that are not reasonable.

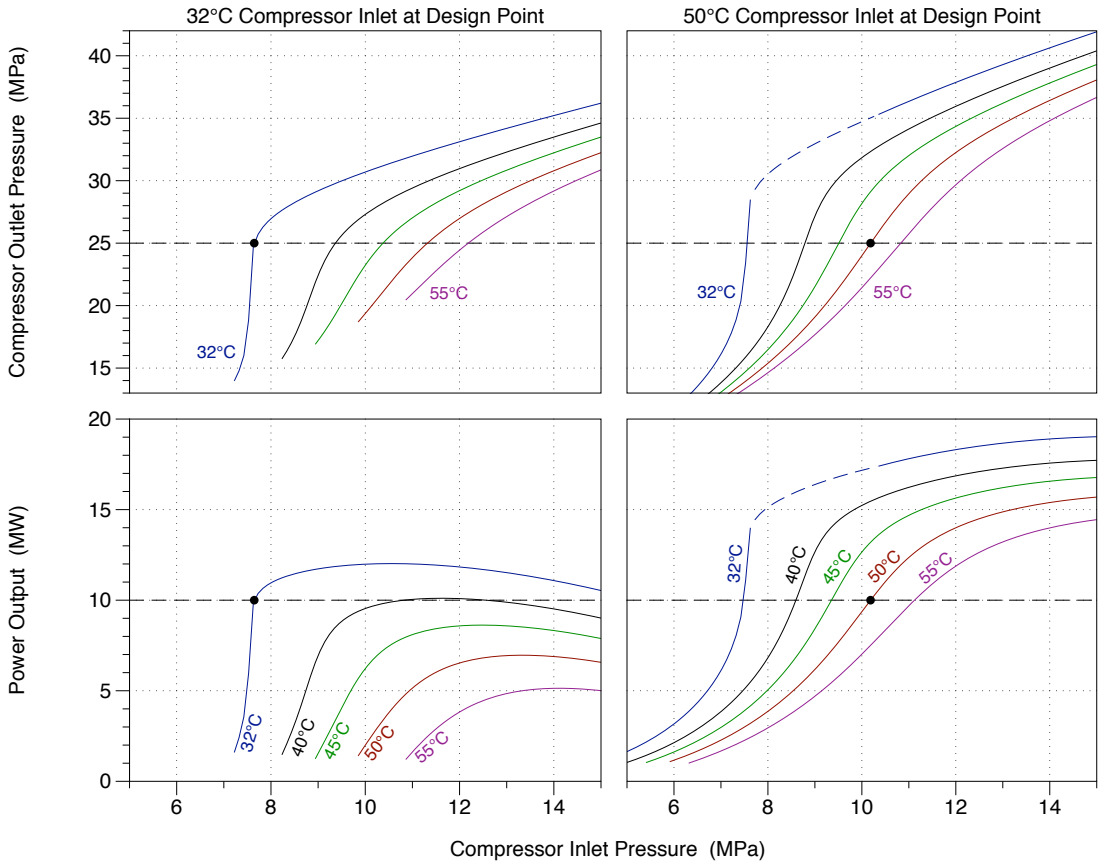


Figure 6.1. Power output and corresponding compressor outlet pressure as a function of compressor inlet pressure for various off-design temperatures. The black dots represent the design points for the two cycles, and the dashed line indicates surge in the main compressor. Note that these results do not take into account physical limitations of the turbomachinery.

The results shown in Fig. 6.1 are a correct representation of the expected off-design high-side pressures if the physical limitations of the system are not considered, but these pressures are beyond what can be reasonably expected for the two designs. In other words, a cycle designed around a 25 MPa high-side pressure will likely not be constructed to withstand pressures in excess of 40 MPa. Besides the high-pressure concerns, the results shown in Fig. 6.1 do not take into account the valid operating envelope of the turbomachinery in the cycle. Compressor surge is not a significant concern for these operating conditions; the dashed line in Fig. 6.1 for the 32°C operating temperature of the 50°C design indicates the only conditions resulting in surge. However, another important consideration in the off-design performance of turbomachinery is the ratio of rotor tip speed to the local speed of sound (referred to in this document as the "tip speed ratio"). Ratios greater than one (1) will result in unstable performance due to shock formation at the tips of the rotors and should be avoided. For this analysis, the determination of the local speed of sound is based on the outlet conditions for the radial compressors and at the inlet conditions for the radial in-flow turbine. While these conditions do not correspond exactly to the conditions at the blades of the turbomachinery, they are a close approximation and sufficient for this analysis. The tip speed ratios corresponding to the operating conditions shown in Fig. 6.1 are depicted in Figure 6.2.

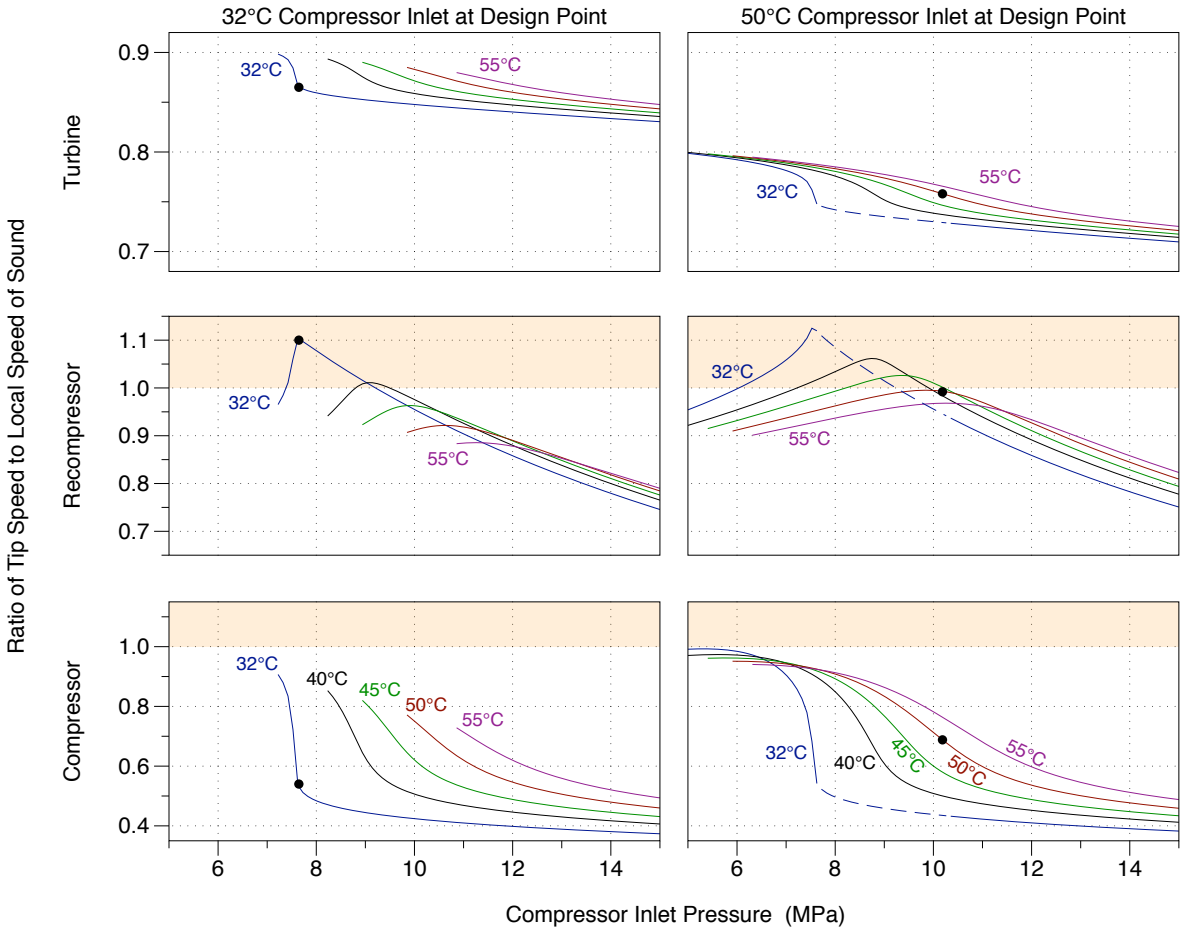


Figure 6.2. Ratio of turbomachinery tip speed to local speed of sound for various off-design low-side temperatures and pressures; values greater than one are not physically valid.

Based on the results shown in Fig. 6.2, the recompressor is responsible for limiting the valid operating envelope of the two representative designs. The reason the recompressor, which is designed using the same non-dimensional head-flow curve as the compressor, has higher tip speed ratio values is because it is compressing carbon dioxide away from the critical point (due to its higher inlet temperatures). Recall that the dimensionless head coefficient of a compressor is its ratio of isentropic enthalpy rise to rotor tip speed squared. The design-point head coefficient for the recompressor and compressor are identical, but the required isentropic enthalpy change of the recompressor is larger than the compressor due to its higher inlet temperature. Therefore, the recompressor requires a larger tip speed.

The unreasonably high pressure attained by the system and the non-valid turbomachinery operating conditions are accounted for in this analysis by setting a high-pressure limit of 30 MPa (while maintaining the 25 MPa design-point compressor outlet pressure) and by using a two-stage recompressor. The two-stage recompressor is modeled by applying the same dimensionless head-flow curve based on the SNL geometry to each stage and is implemented in the file `snl_compressor_tsr.f90`. The two-stage recompressor can also be considered as two single-stage compressors in series on a common shaft, without intercooling. Setting a high-pressure limit and modeling a two-stage recompressor results in the allowable power output and tip speed ratios shown in Figures 6.3 and 6.4.

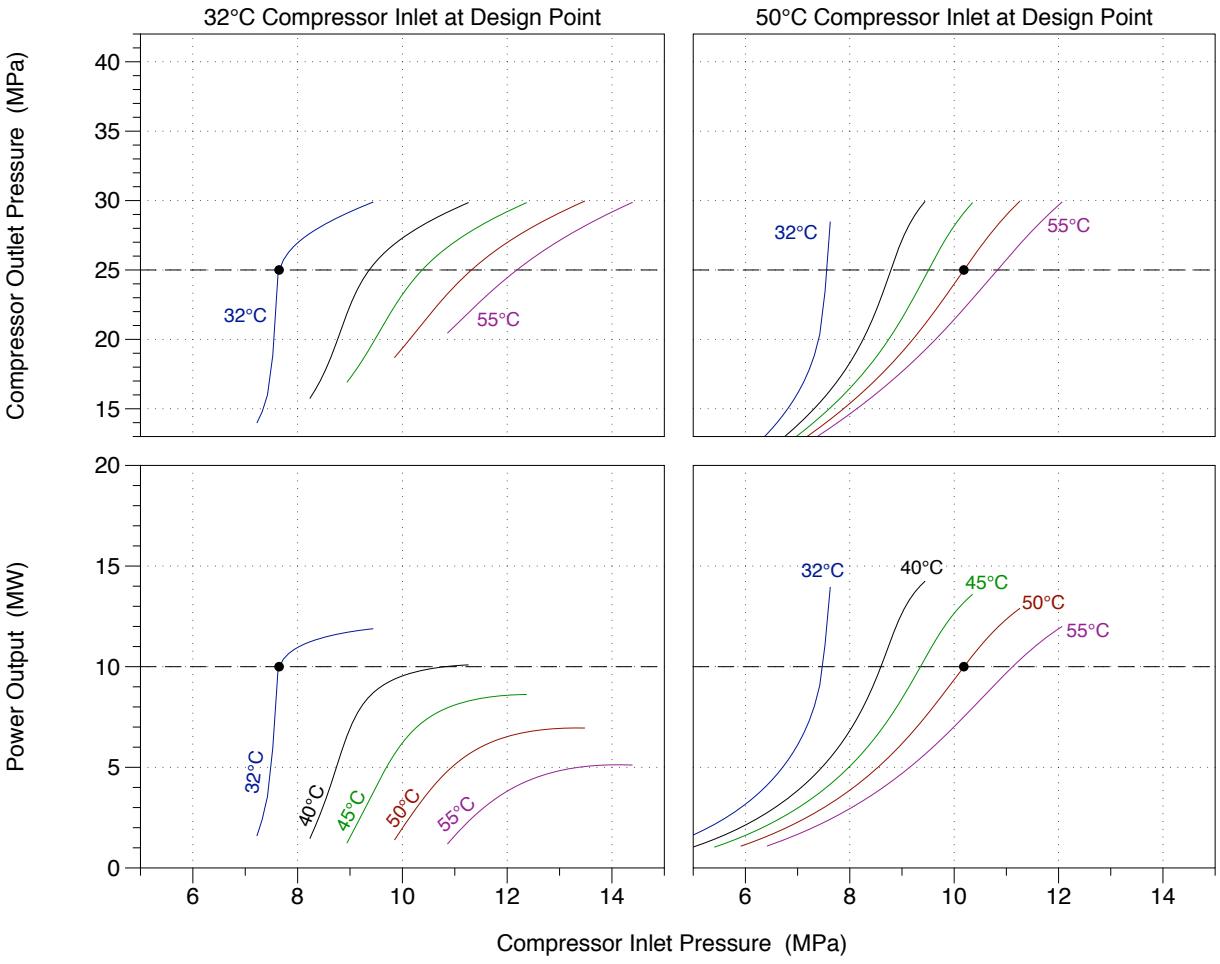


Figure 6.3. Power output and compressor outlet pressure for various off-design conditions. Only valid operating points are plotted.

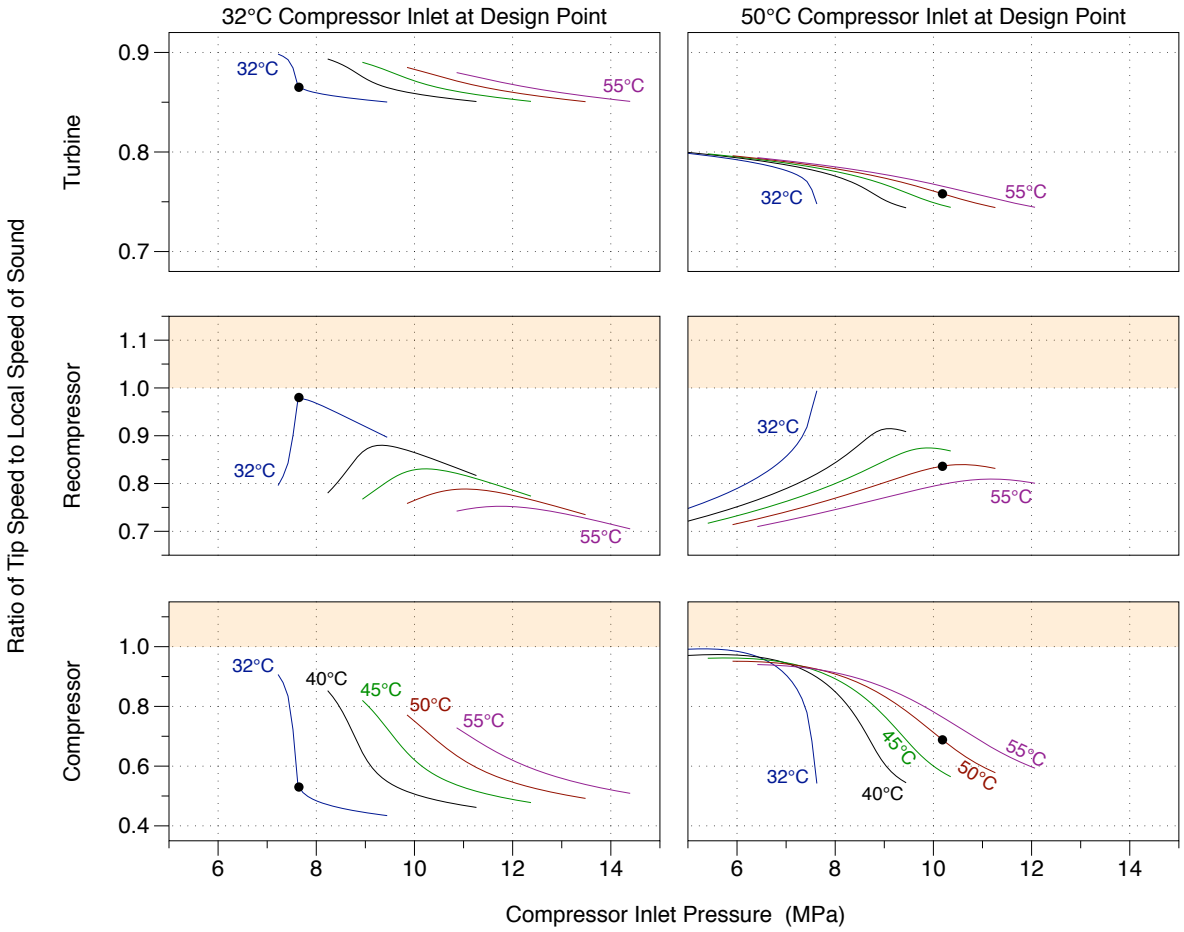


Figure 6.4. Ratio of turbomachinery tip speed to local speed of sound for various off-design low-side temperatures and pressures.

The predicted power output shown in Figure 6.3 is calculated with the main shaft speed and recompression fraction fixed at their design-point values. In order to investigate the relationship between the optimal off-design performance of the two designs, as well as to compare the performance the recompression cycle to the simple recuperated cycle, four cycle designs are considered: low-temperature recompression, high-temperature recompression, low-temperature simple, and high-temperature simple. The four designs are constrained to generate 10 MW of mechanical power output, excluding generator losses and power associated with heat rejection and addition, at the design point with a high-side (turbine inlet) temperature of 550°C and a high-side pressure of 25 MPa. The recompression cycles assume a total recuperator conductance of 3,000 kW/K between the two heat exchangers, while the simple cycles assume a single recuperator with a conductance of 1,500 kW/K. These values are chosen to balance cycle perfor-

mance and heat exchanger size. For the recompression cycles, the distribution of conductance is optimized at the design point. Pressure drops through the heat exchangers, including the precooler and primary heat exchanger, are assumed to be 1 percent (relative to the inlet pressure). The design-point compressor, recompressor, and turbine isentropic efficiencies are 0.89, 0.89, and 0.93, respectively, which are suggested by Dostal (2004) as appropriate targets for SCO_2 power cycles. Note that the recompressor isentropic efficiency is the overall efficiency, which requires a stage efficiency of approximately 0.896. The parameters for the four designs of interest are summarized in Table 6.1, as well as the resulting design-point thermal efficiencies. The thermal efficiency of the cycle does not take into account losses associated with the electric generator or any balance of plant required power (e.g., auxiliary pumping or fan power for heat addition or heat rejection from the cycle). The distribution of total conductance and recompression fraction for the recompression designs is optimized using the `optimal_design` subroutine discussed previously.

Table 6.1. Four designs of interest.

	Low-Temperature Design		**High-Temperature Design**	
	Simple	**Recompression**	**Simple**	**Recompression**
Power Output	10 MW			
Turbine Inlet Temperature	550°C			
Compressor Outlet Pressure	25 MPa			
Compressor Isentropic Efficiency	0.89			
Turbine Isentropic Efficiency	0.93			
Heat Exchanger Pressure Drops	1%			
Compressor Inlet Temperature	32°C		50°C	
Compressor Inlet Pressure	8 MPa	8 MPa	9 MPa	10 MPa
LT Recuperator Conductance	1.5 MW/K	1.7 MW/K	1.5 MW/K	1.5 MW/K
LT Recuperator Minimum ΔT	1.4°C	5.2°C	3.7°C	7.2°C
LT Recuperator Approx. Volume	60 m^3	81 m^3	38 m^3	37 m^3
HT Recuperator Conductance	-	1.3 MW/K	-	1.5 MW/K
HT Recuperator Minimum ΔT	-	5.0°C	-	11.4°C
HT Recuperator Approx. Volume	-	38 m^3	-	34 m^3
Compressor Rotor Diameter	0.138 m	0.118 m	0.213 m	0.183 m
RC First Stage Rotor Diameter	-	0.162 m	-	0.157 m
RC Second Stage Rotor Diameter	-	0.138 m	-	0.139 m
Turbine Rotor Diameter	0.242 m	0.205 m	0.272 m	0.251 m
Turbine Effective Nozzle Area	2,340 mm^2	2,790 mm^2	3,090 mm^2	3,500 mm^2
Main Shaft Speed	31,410 rpm	36,670 rpm	26,580 rpm	27,040 rpm
Recompressor Shaft Speed	-	33,250 rpm	-	32,750 rpm
Recompression Fraction	-	0.383	-	0.258
Turbine Mass Flow Rate	82.8 kg/s	98.5 kg/s	114.3 kg/s	134.2 kg/s
Thermal Efficiency	41.6 %	47.4 %	38.8 %	41.8 %

The compressor inlet pressure of 8 MPa for the two low-temperature designs is slightly larger than the optimal design-point low-side pressures of 7.65 MPa for the recompression cycle and 7.54 MPa for the simple cycle. The decrease in design-point thermal efficiency associated with increasing these values to 8 MPa is approximately 0.5 and 0.3 percentage points for the simple and recompression cycles, respectively, and moving slightly away from the critical pressure increases the numerical stability of the

model. The approximate volume of the recuperators is determined by assuming a counterflow configuration with flow channels that are 5 mm wide and 2.5 mm deep, which is representative of the printed circuit heat exchangers (PCHE) being considered for use with SCO_2 power cycles (Dostal, 2004). At the design point the simple cycle has a lower thermal efficiency than the recompression cycle, but this disadvantage is balanced by the advantage of requiring one less heat exchanger and one less compressor. Interestingly, the thermal efficiency advantage of the recompression cycle is not as significant for the high-temperature designs. The reason for this difference is because the high-temperature designs operate further from the critical region of CO_2, decreasing the heat capacitance imbalance in the recuperators and reducing the necessity for the recompressor. This effect is also apparent by the decreased recompression fraction of the high-temperature recompression design compared to the low-temperature recompression design. The selection of an optimal design and design point will ultimately depend upon application-specific economics, but it is interesting to note that the efficiency of the high-temperature simple design is within 3 percentage points of the high-temperature recompression design and does not require a second recuperator or compressor. The four designs described in Table 6.1 are used to explore the effects of off-design compressor and turbine inlet temperatures in the following sections.

6.1. Varying Compressor Inlet Temperature

The effect of compressor inlet temperature on cycle performance is non-trivial. Figure 6.5 is a plot of the predicted power output and thermal efficiency for the low-temperature and high-temperature recompression designs as the compressor inlet pressure varies for three compressor inlet temperature cases (32°C, 40°C, and 50°C). Figure 6.6 is a similar plot for the designs based on the simple cycle configuration. At each point, the main shaft speed and recompression fraction are either fixed (dotted lines), varied to maximize efficiency (solid lines), or varied to maximize power output of the cycle (dashed lines).

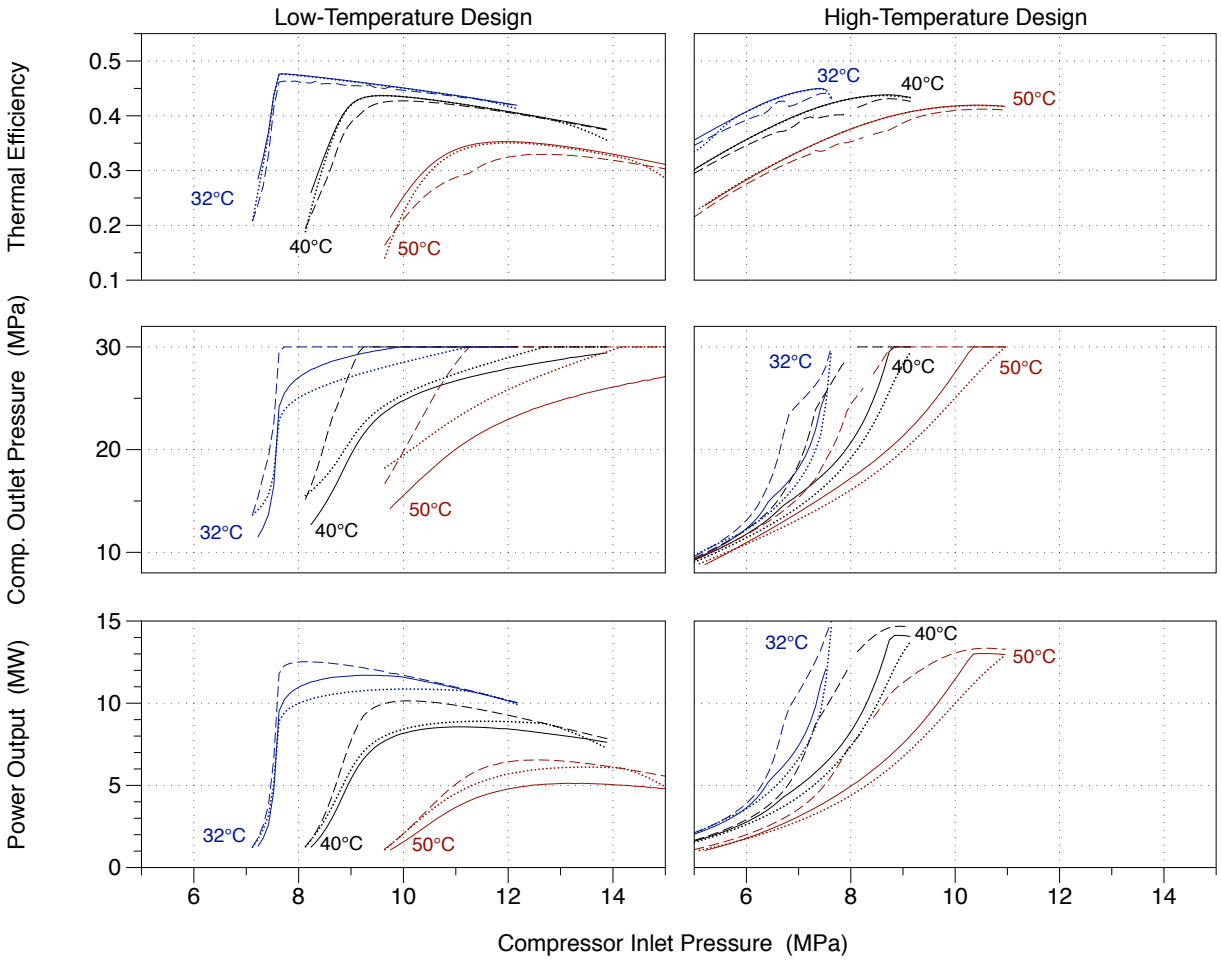


Figure 6.5. Predicted power output, high-side pressure, and thermal efficiency for the low-temperature and high-temperature recompression designs. The solid lines indicate operation at maximum efficiency, the dashed lines indicate operation at maximum power output, and the dotted lines are operation at maximum efficiency using design-point shaft speed.

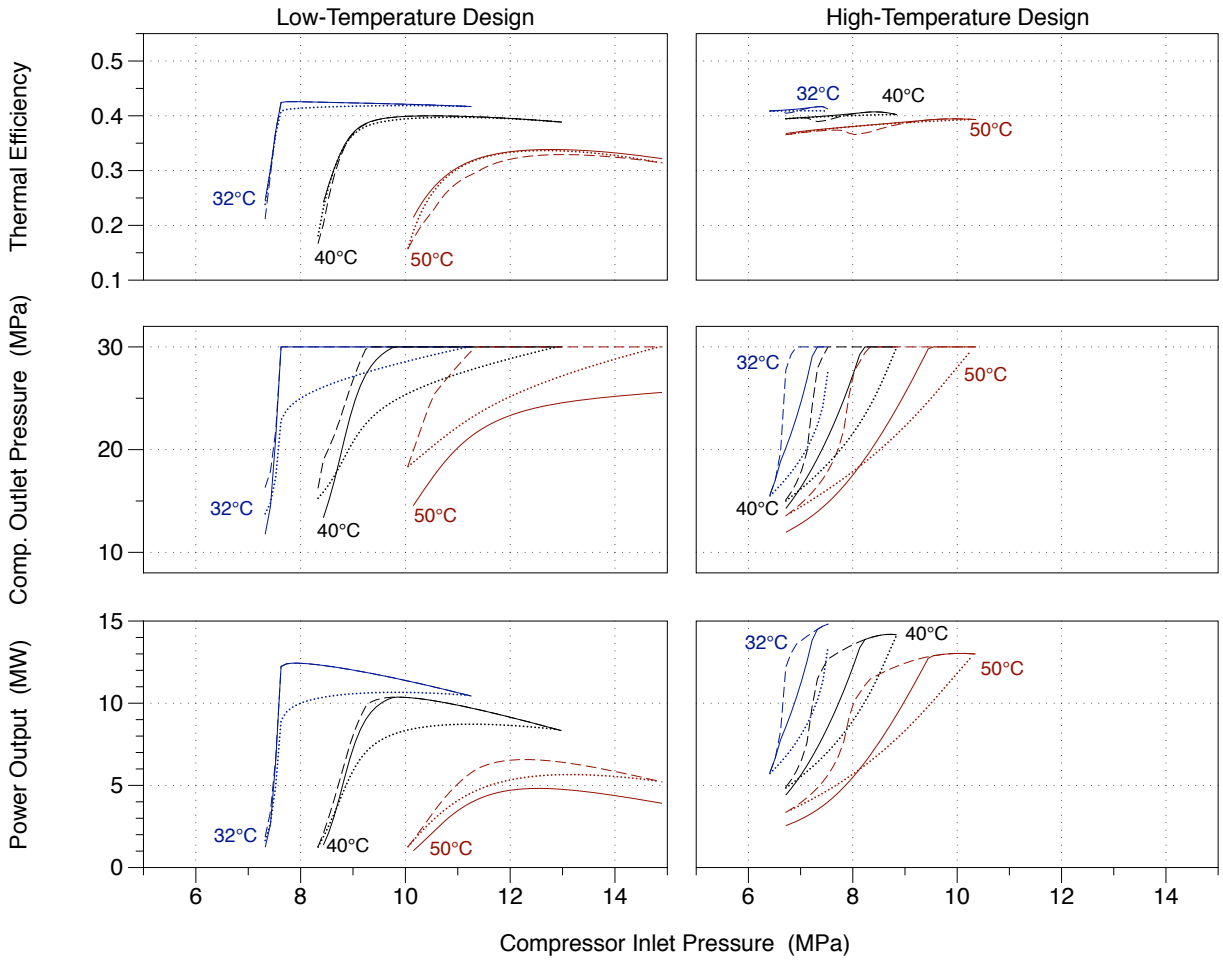


Figure 6.6. Predicted power output, high-side pressure, and thermal efficiency for the low-temperature and high-temperature simple designs. The solid lines indicate operation at maximum efficiency, the dashed lines indicate operation at maximum power output, and the dotted lines are operation at design-point shaft speed.

Figures 6.5 and 6.6 reveal a number of interesting operating characteristics of the four designs. One important conclusion is that, for the low-temperature designs, there are power output values that cannot be attained at higher off-design compressor inlet temperatures. For example, the maximum possible power output of the low-temperature recompression design at 50°C is 6.5 MW. Therefore, that design is incapable of producing the rated 10 MW of power output under those conditions. However, the high-temperature designs do not show this limitation and can operate over a wider range of temperatures at the rated 10 MW power output. The reason why the high-temperature designs are not as limited is because, for this analysis, their off-design operation occurs at temperatures below the design point. At lower tempera-

tures the density of carbon dioxide increases, allowing for increased mass flow rate and power production of the cycle.

The optimal (with respect to efficiency) performance of the four designs over a range of compressor inlet temperatures is predicted in Figure 6.7. Three power outputs are considered: 10 MW (the rated power output of the cycle), 7.5 MW, and 5 MW. Results are only shown if the design is capable of achieving the target power output. Note that these results are generated using the more conservative low-reaction radial turbine model (as opposed to the modified, SNL radial turbine model).

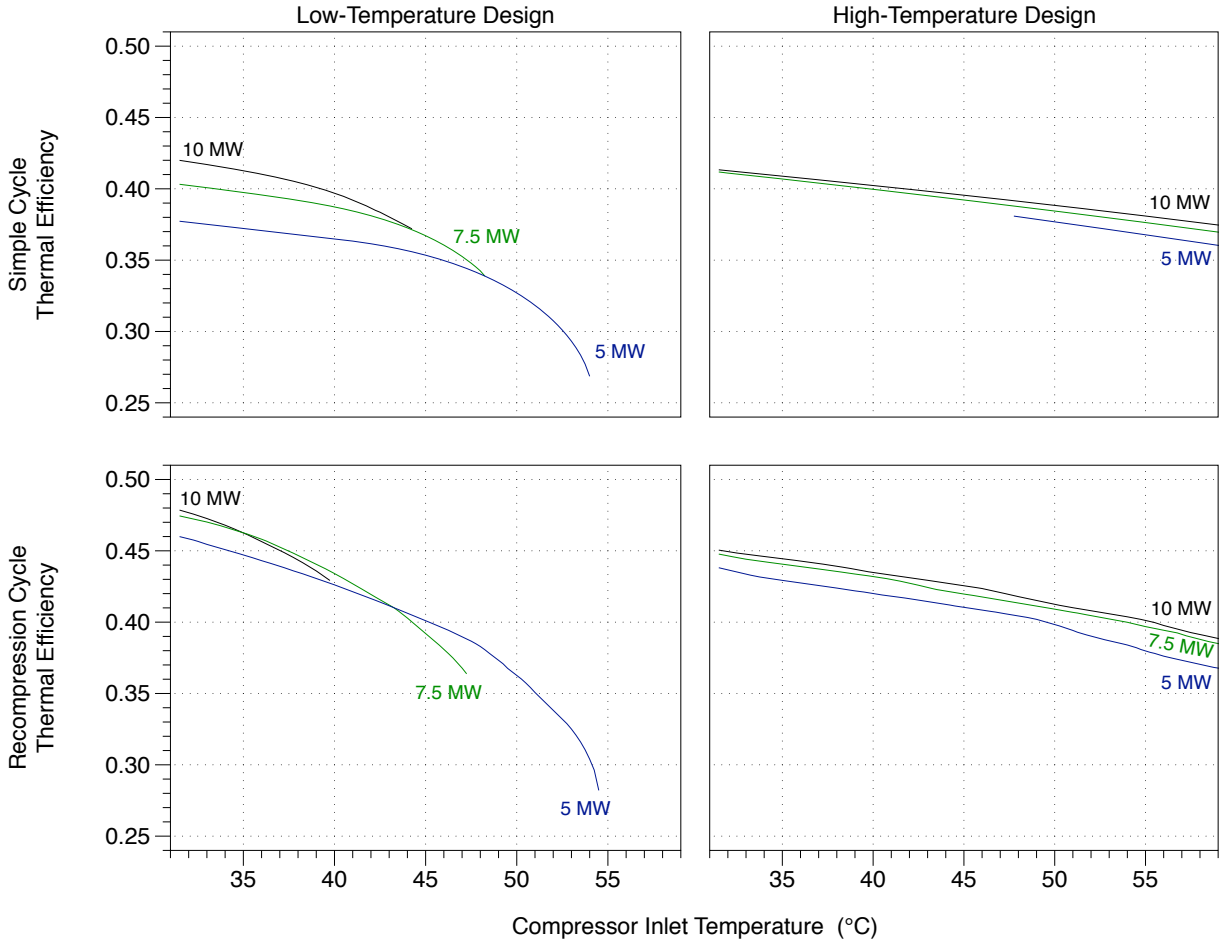


Figure 6.7. Thermal efficiency of the four designs as a function of off-design compressor inlet temperature.

The low-temperature designs are limited with respect to operation at the rated power output under off-design conditions, but the high-temperature designs due not share this limitation. The high-tempera-

ture simple design does show a limitation with respect to part-load operation at 5 MW under cooler off-design temperatures, but this does not take into alternative part-load strategies (e.g., a turbine bypass valve) that are not included in this analysis. Note that the design-point efficiencies of the low-temperature designs at 32°C are 41.6 and 47.4 percent for the simple and recompression configurations, respectively, while the off-design efficiencies of the high-temperature designs operating at 32°C are 41.3 and 44.9 percent. This result is significant, as it strongly suggests that a higher temperature design is preferred over a lower temperature design, especially when considering simple cycles. A case can be made for the low-temperature recompression design if it will never experience off-design operation, as the 2.5 percentage point increase in efficiency is significant. For power cycles expected to operate under off-design conditions, however, the results in Fig. 6.7 clearly show the advantage of designing for a higher temperature and taking advantage of lower temperature off-design conditions.

The control parameters corresponding to the efficiencies plotted in Fig. 6.7 are shown in Figure 6.8 for the recompression cycle configurations and Figure 6.9 for the simple cycle configurations. The optimal control variables are not always represented as smooth curves due to the numerical noise associated with the optimization algorithms and tolerances chosen for this analysis.

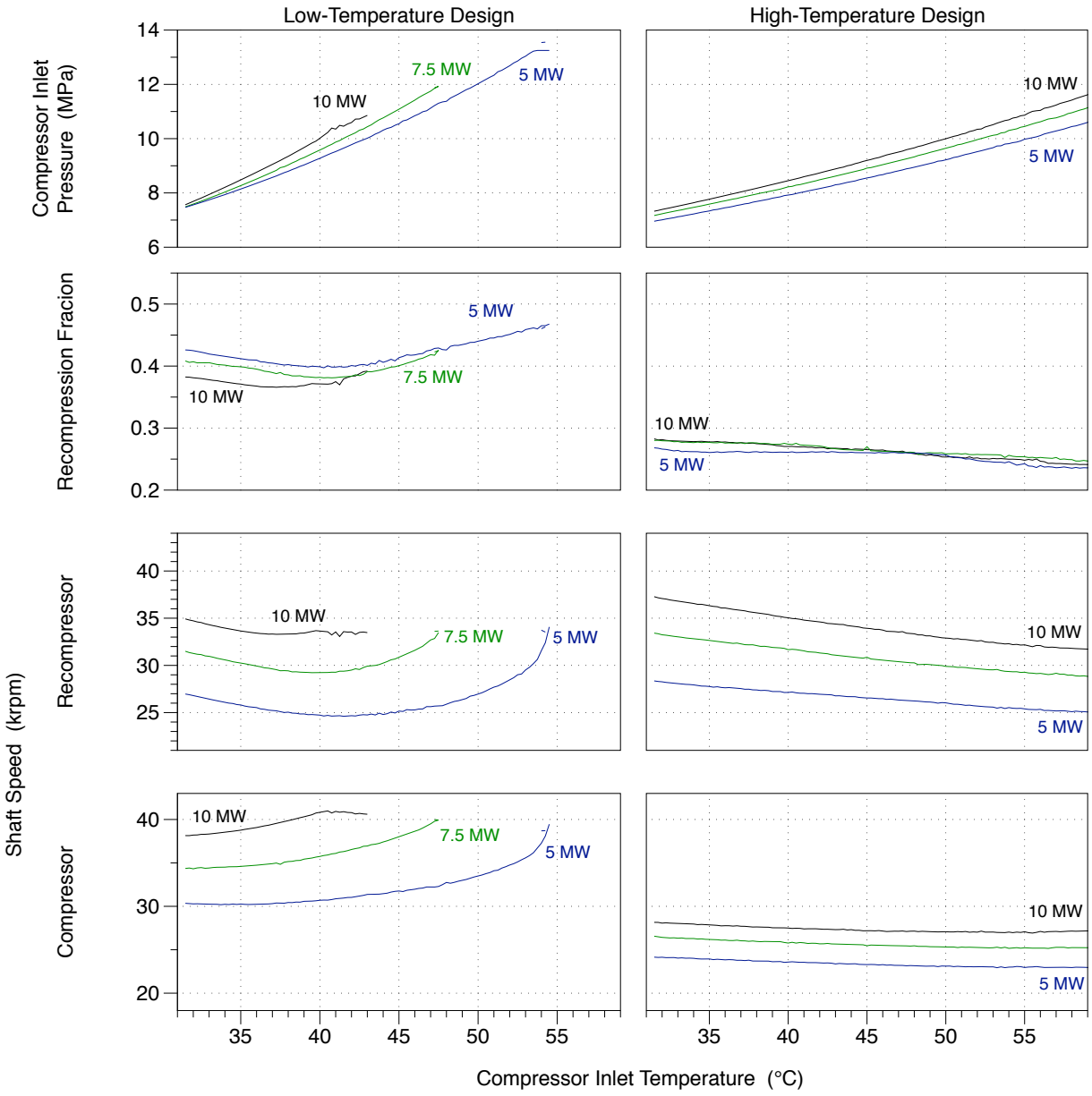


Figure 6.8. Control parameters associated with optimal performance under off-design compressor inlet temperature for the two recompression cycle designs.

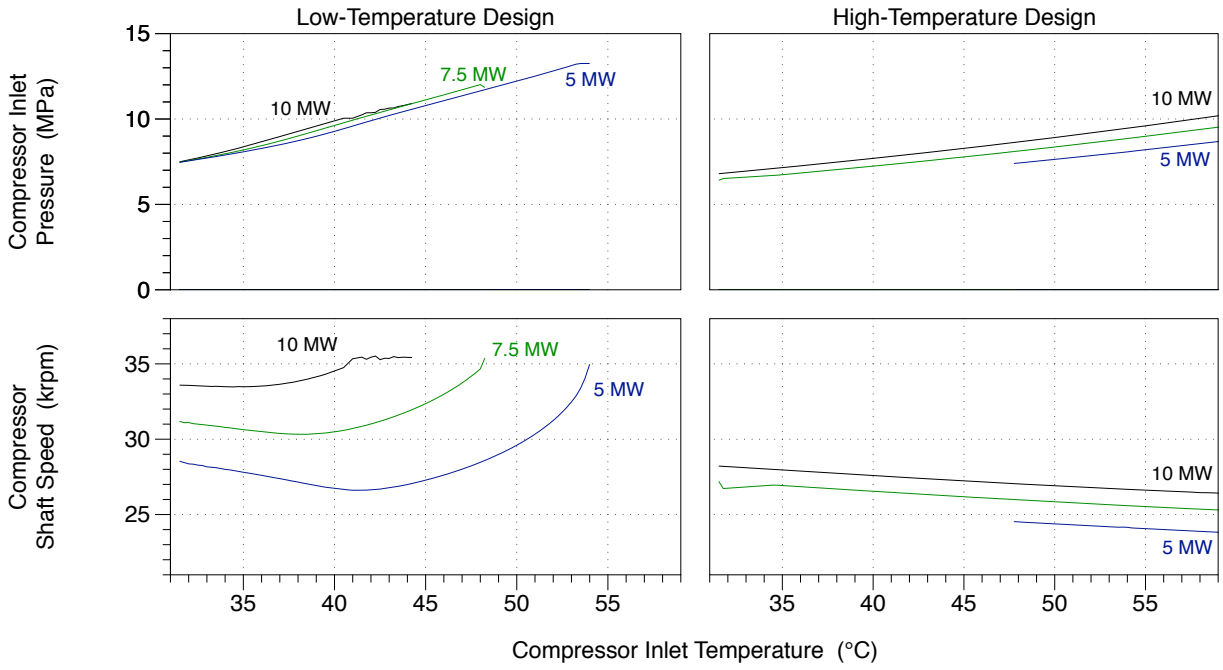


Figure 6.9. Control parameters associated with optimal performance under off-design compressor inlet temperature for the two simple cycle designs.

Figures 6.8 and 6.9 highlight the importance of inventory control while operating under off-design compressor inlet temperatures in order to maintain a target power output while operating at the highest thermal efficiency. The optimal performance of the cycles under off-design conditions results from a balance of the optimal pressure ratio of the cycle (from a thermodynamic sense) and the off-design efficiencies of the turbomachinery. The head coefficient (ϕ) of the compressors and the ratio of tip speed to spouting velocity (ν) for the turbine that corresponds to the efficiencies plotted in Fig. 6.7 are shown in Figure 6.10 for the recompression designs and Figure 6.11 for the simple designs. The compressor flow coefficients do not deviate significantly from the design-point value of 0.0297. However, note that the turbine ν values do deviate from the design-point value of 0.707 and values near 0.9 are possible. At these higher values the conservative efficiency curve of the low-reaction radial turbine model has a significant effect on the predicted performance of the cycles. The performance of similar cycles modeled with the higher-performance SNL radial turbine model is explored in Section 6.3.

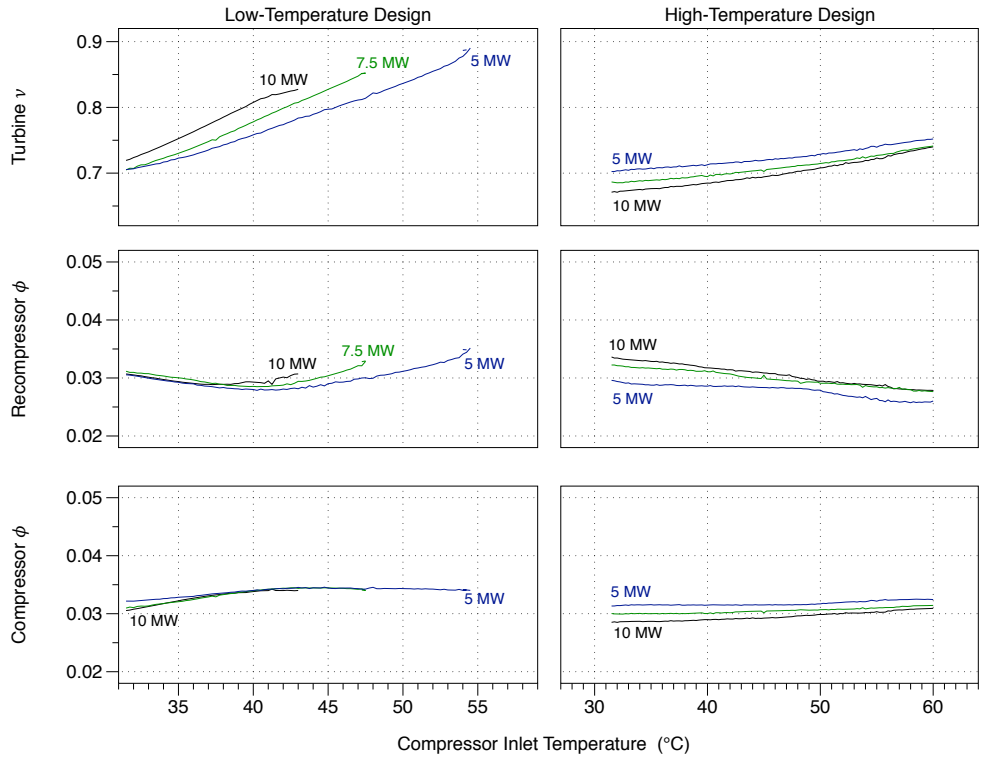


Figure 6.10. Head coefficient for the compressor and recompressor and turbine spouting velocity ratio under off-design conditions for the recompression designs.

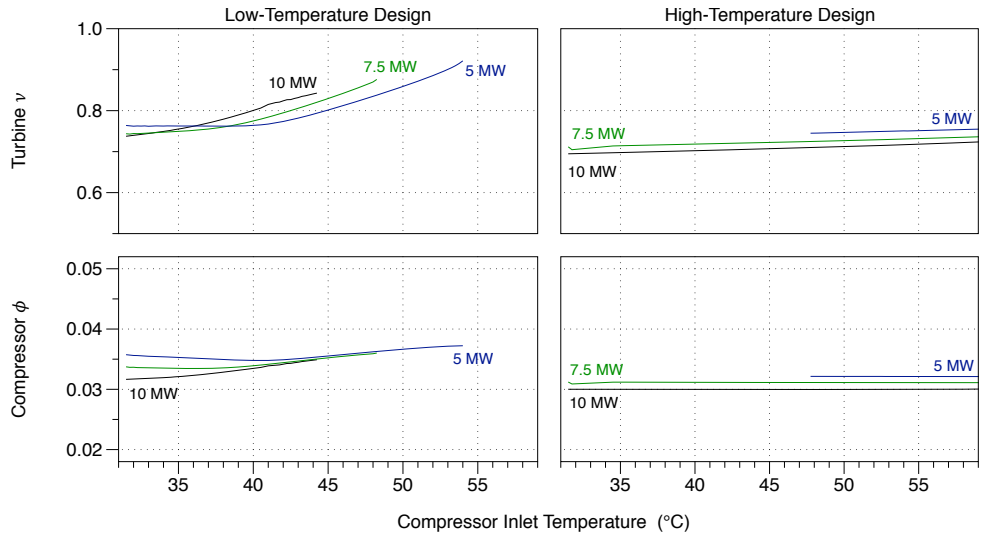


Figure 6.11. Head coefficient for the compressor and turbine spouting velocity ratio under off-design conditions for the simple designs.

6.2. Varying Turbine Inlet Temperature

Compared to operation under off-design compressor inlet temperatures, the effect of off-design turbine inlet temperature is relatively uneventful. Figure 6.13 is a plot of the predicted power output and thermal efficiency for the recompression designs as the compressor inlet pressure varies for three turbine inlet temperature. Figure 6.13 is a similar plot for the designs based on the simple cycle configuration.

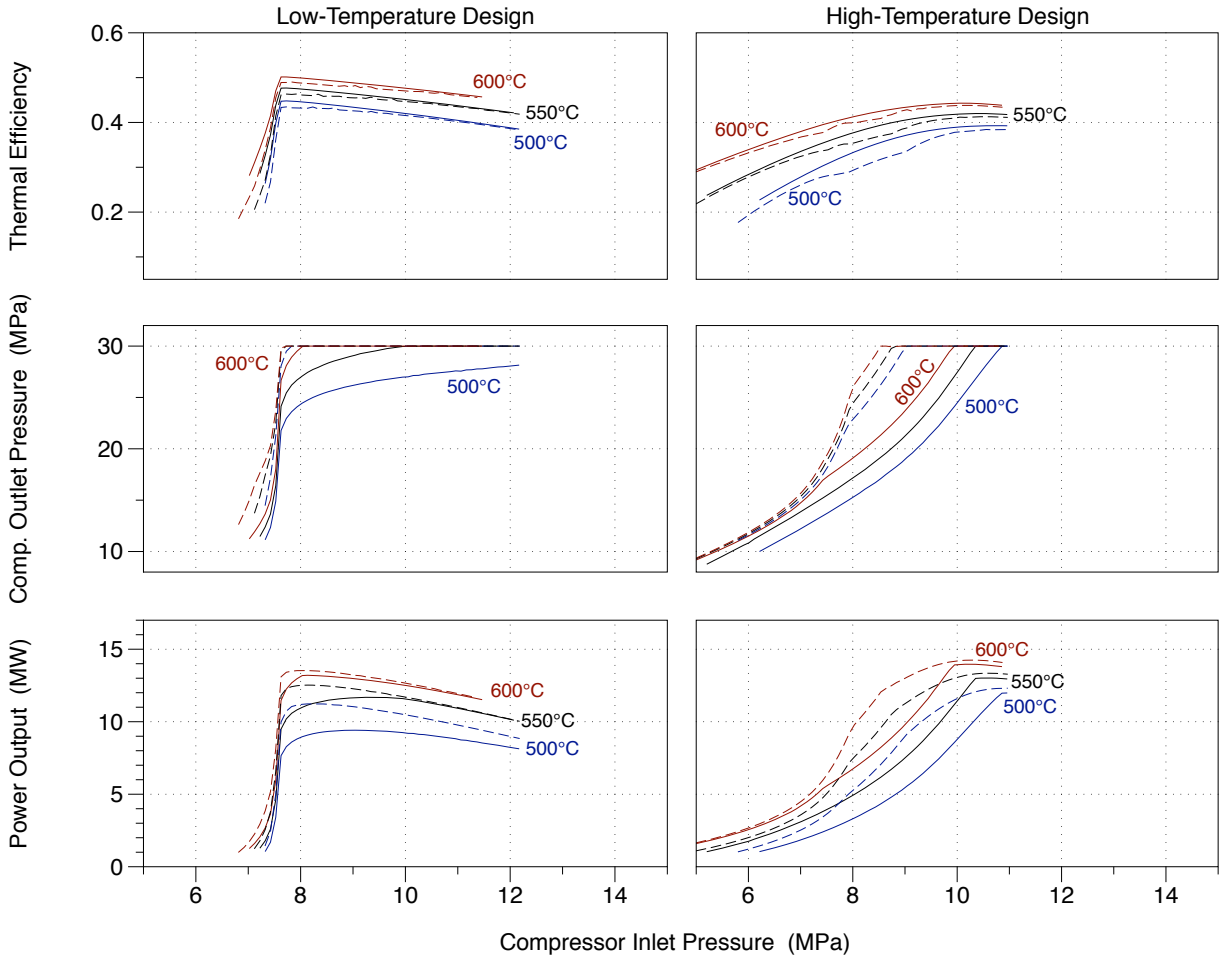


Figure 6.12. Predicted power output, high-side pressure, and thermal efficiency for the low-temperature and high-temperature recompression designs. The solid lines indicate operation at maximum efficiency, the dashed lines indicate operation at maximum power output, and the dotted lines are operation at maximum efficiency using design-point shaft speed.

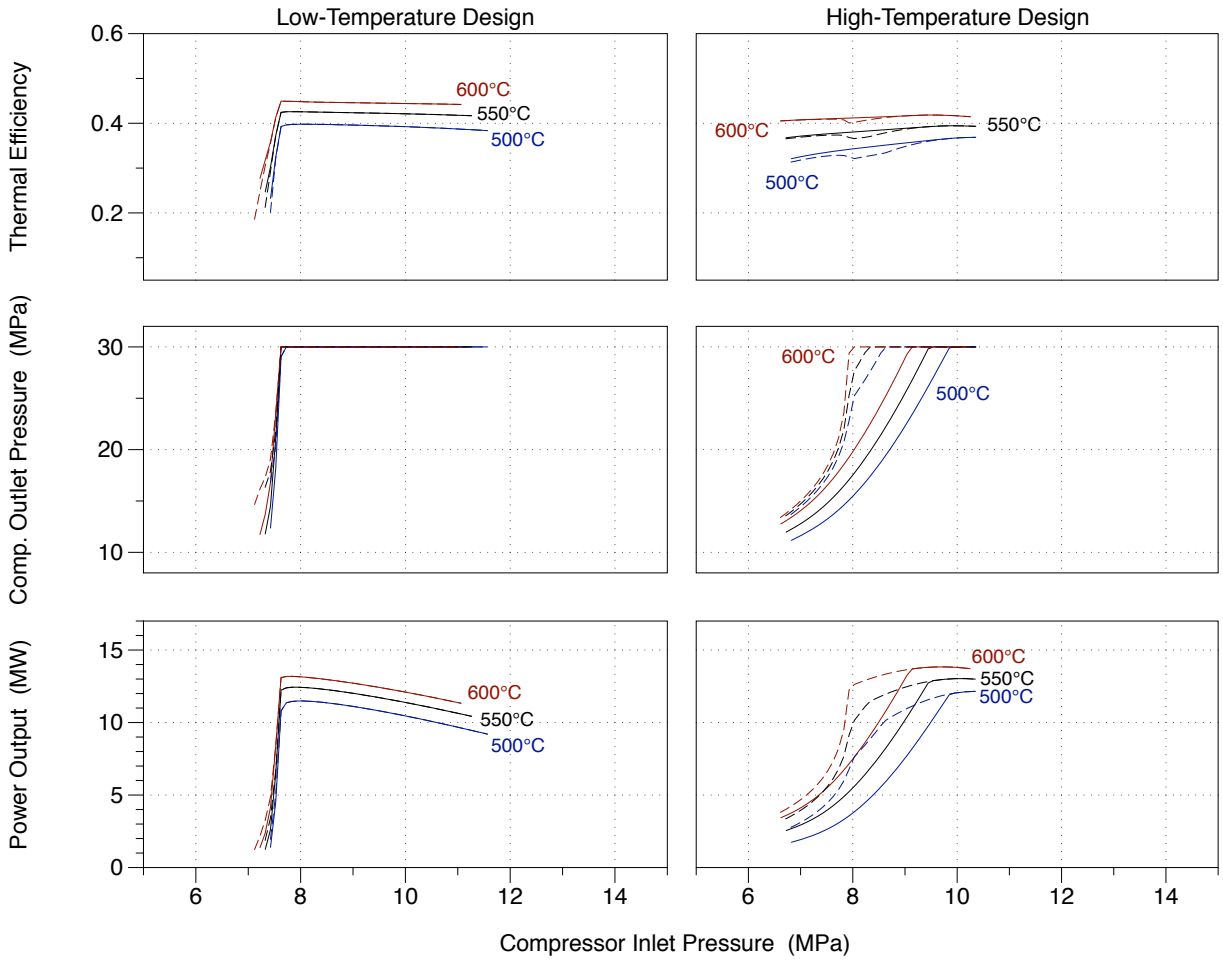


Figure 6.13. Predicted power output, high-side pressure, and thermal efficiency for the low-temperature and high-temperature simple designs. The solid lines indicate operation at maximum efficiency, the dashed lines indicate operation at maximum power output, and the dotted lines are operation at design-point shaft speed.

Increasing turbine inlet temperature increases the power output and thermal efficiency of the cycle, but the overall impact on cycle operation is minimal. Note that changes to the off-design turbine inlet temperature do not result in unobtainable power outputs, as is the case for changes to the off-design compressor inlet temperature. This result is also apparent in Figure 6.14, which is the maximum thermal efficiency of the four designs over a range of off-design turbine inlet temperatures.

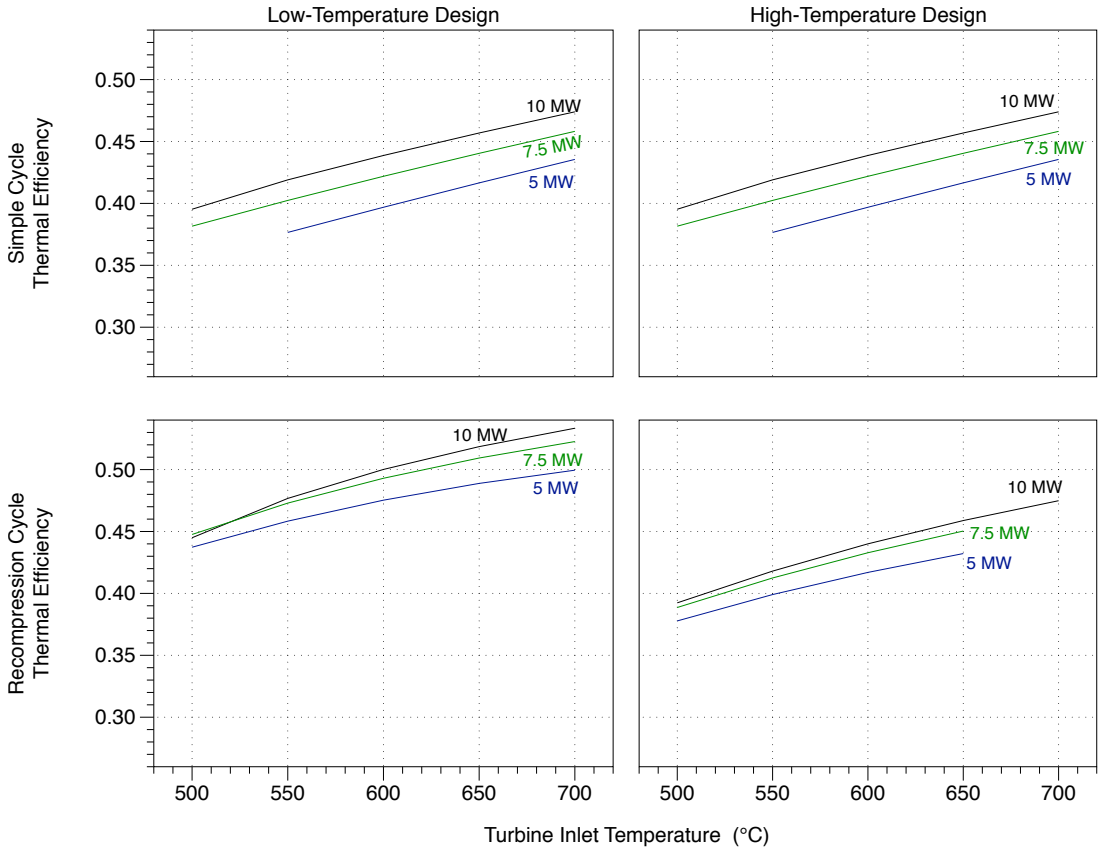


Figure 6.14. Thermal efficiency of the four designs as a function of off-design turbine inlet temperature.

6.3. Cycle Performance with Alternative Turbine Model

The alternative turbine model that is based on the SNL radial turbine efficiency curve and implemented in the file `snl_radial_turbine.f90` is used to explore the off-design performance of three recompression cycle designs. The three designs correspond to the single-shaft recompression configuration shown in Fig. 3.15(a) and are characterized by their compressor inlet temperatures of 32°C, 40°C, and 50°C, respectively. Similar to the cycles discussed in Section 6.1, each design is sized to generate 10 MW of mechanical power output at the design point, excluding generator losses and power associated with heat rejection and addition, with a high-side (turbine inlet) temperature of 550°C and a high-side pressure of 25 MPa. The total recuperator conductance at the design point is 3,000 kW/K. Pressure drops through the heat exchangers, including the precooler and primary heat exchanger, are again assumed to be 1 per-

cent. The design-point compressor, recompressor, and turbine isentropic efficiencies are 0.89, 0.89, and 0.93, respectively. The parameters for the three designs are summarized in Table 6.2, as well as the resulting design-point thermal efficiencies. The thermal efficiency of the cycle does not take into account losses associated with the electric generator or any balance of plant required power (e.g., auxiliary pumping or fan power for heat addition or heat rejection from the cycle). The compressor inlet pressure, distribution of total conductance, and recompression fraction is optimized using the `optimal_design` subroutine contained in the `design_point` module. The approximate volume of the recuperators is again determined by assuming a counter-flow configuration with flow channels that are 5 mm wide and 2.5 mm deep.

Table 6.2. Three designs of interest, characterized by their compressor inlet temperatures.

Power Output	10 MW		
Turbine Inlet Temperature	550°C		
Compressor Outlet Pressure	25 MPa		
Compressor Isentropic Efficiency	0.89		
Turbine Isentropic Efficiency	0.93		
Heat Exchanger Pressure Drops	1%		
Compressor Inlet Temperature	32°C	40°C	50°C
Compressor Inlet Pressure	7.7 MPa	9 MPa	10 MPa
LT Recuperator Conductance	1.74 MW/K	1.59 MW/K	1.52 MW/K
LT Recuperator Minimum ΔT	5.3°C	7.2°C	7.2°C
LT Recuperator Approx. Volume	80 m^3	50 m^3	40 m^3
HT Recuperator Conductance	1.26 MW/K	1.41 MW/K	1.48 MW/K
HT Recuperator Minimum ΔT	5.1°C	7.7°C	11.4°C
HT Recuperator Approx. Volume	40 m^3	35 m^3	35 m^3
Compressor Rotor Diameter	0.120 m	0.148 m	0.183 m
RC First Stage Rotor Diameter	0.162 m	0.162 m	0.157 m
RC Second Stage Rotor Diameter	0.137 m	0.141 m	0.139 m
Turbine Rotor Diameter	0.218 m	0.241 m	0.265 m
Turbine Effective Nozzle Area	1,140 mm^2	1,450 mm^2	1,790 mm^2
Main Shaft Speed	37,080 rpm	31,410 rpm	27,030 rpm
Recompressor Shaft Speed	34,620 rpm	32,570 rpm	32,790 rpm
Recompression Fraction	0.3752	0.3266	0.2578
Turbine Mass Flow Rate	96.8 kg/s	114.5 kg/s	134.2 kg/s
Thermal Efficiency	47.7 %	45.0 %	41.8 %

The primary difference between the three designs is the size of the turbomachinery, specifically the main compressor and turbine; designing for warmer compressor inlet temperatures results in slightly larger turbomachinery operating at lower shaft speeds. The recompressor design does not follow this trend as its size is driven by the optimal recompression fraction, which decreases as the low-side temperature increases. The warmer designs require less recompression because the properties of carbon dioxide do not vary as rapidly at conditions away from the critical point, decreasing the amount of flow that must be diverted in order to balance the hot and cold stream capacitance rates of the recuperator. Note that the effective nozzle areas for these designs are significantly smaller than the effective nozzle areas reported in Table 6.1 for similar designs. The reason for this difference is the alternative turbine model, which uses the density at the turbine inlet, rather than outlet, when sizing the turbine. The inlet density is larger, therefore the required effective nozzle area is smaller for a comparable mass flow rate.

The off-design thermal efficiency at the rated power output is plotted in Figure 6.15 for the three designs, and the corresponding control parameters are plotted in Figure 6.16. Note that the efficiency degradation associated with the low-temperature (32°C) design operating at warmer low-side temperatures is not as severe as predicted in the analysis presented in Section 6.1. These results are consistent with the differences between the two turbine models; namely, the higher efficiency of the SNL turbine model at larger ν values.

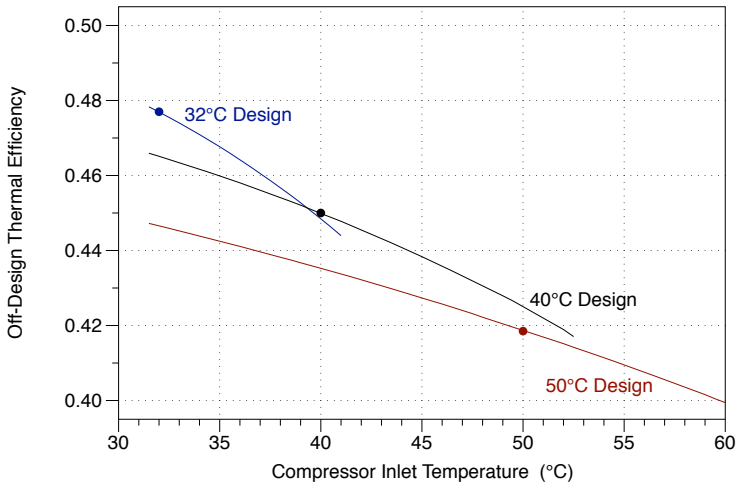


Figure 6.15. Thermal efficiency of the three designs at the rated 10 MW power output as a function of off-design compressor inlet temperature. The design points are indicated with a circle and only achievable values (with a high-pressure limit of 30 MPa) are plotted.

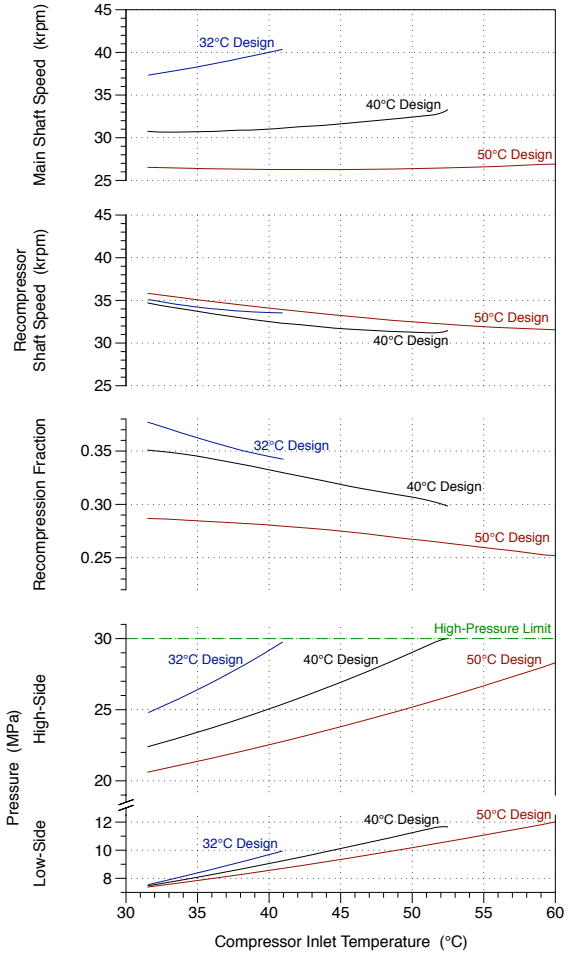


Figure 6.16. Control parameters associated with the efficiencies predicted in Fig. 6.15.

Three shaft configurations are considered for the 32°C design and 50°C design described in Table 6.2. In the "Normal" configuration, a single shaft connects the main compressor and turbine and shaft speed is allowed to vary. The "Fixed-Shaft" configuration is identical to the "Normal" configuration, with the exception that the shaft speed is fixed at its design-point value. In the "Split-Shaft" design, the main compressor and turbine use separate shafts, with the compressor shaft speed allowed to vary and the turbine shaft speed fixed at 3,600 rpm to facilitate the use of a grid-tied, synchronous generator. Note that this configuration does change the geometry of the turbine, while the "Normal" and "Fixed-Shaft" turbine geometries for the two designs remain as listed in Table 6.2. For the "Split-Shaft" configuration, the turbine rotor diameter is 2.25 m for the 32°C design and 2.0 m for the 50°C design. An independent motor with variable speed control drives the recompressor for all three configurations.

The predicted off-design thermal efficiencies at the rated 10 MW power output for the six designs are shown in Figure 6.17. The more complicated "Split-Shaft" configuration performs better under off-design conditions, but for the 50°C design its advantage is minimal. However, these results do not take into account the additional losses associated with the efficiency of the motor used to drive the main compressor. The efficiencies plotted in Fig. 6.17 also do not consider the advantages associated with using a fixed-speed synchronous generator compared to the power electronics that are necessary to condition the power generated using a variable shaft speed in the "Normal" configuration. The operating envelope of the "Fixed-Shaft" configuration for the 32°C design is reduced because that configuration has one less control parameter than the other configurations. The performance of the "Fixed-Shaft" configuration for the 50°C design is not noticeably different than the "Normal" configuration. This result is expected based on the relatively constant optimal off-design shaft speed for that design shown in Fig. 6.16. The thermal efficiency from 10% to 100% of rated power output for the six designs is shown in Figure 6.18, with the compressor and turbine inlet temperatures held constant at their design-point values. Interestingly, the part-load thermal efficiency of the "Normal" configuration is consistently higher than the "Split-Shaft" configuration.

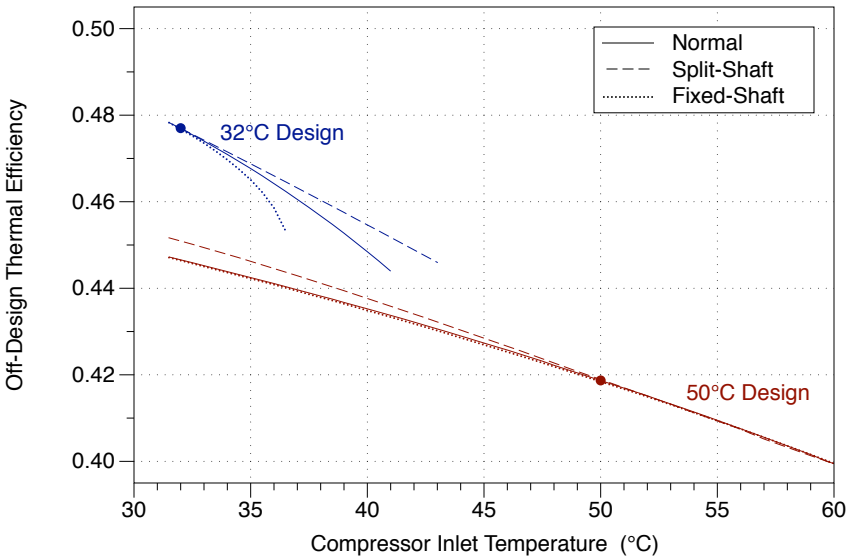


Figure 6.17. Off-design thermal efficiency of the three shaft configurations for the 32°C and 50°C designs at the rated 10 MW power output. The design points are indicated with a circle and only achievable values (with a high-pressure limit of 30 MPa) are plotted.

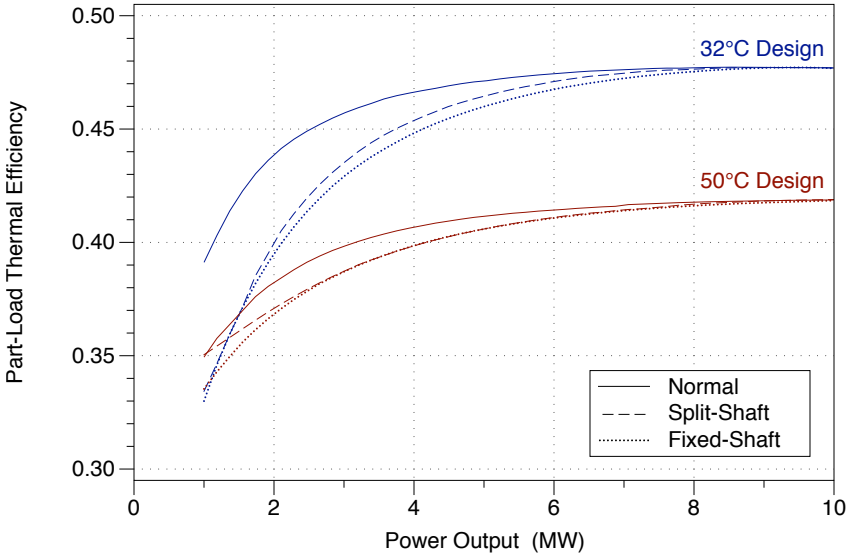


Figure 6.18. Part-load thermal efficiency of the three shaft configurations for the 32°C and 50°C designs.

6.4. Convergence Verification

The data presented in this chapter were generated using convergence tolerance of 10^{-5}, 10 sub-heat exchangers per recuperator for the recompression designs, and 20 sub-heat exchangers for the recuperator in each of the simple designs. More sub-heat exchangers are required for the simple design because the temperature range is not split between two heat exchangers, as it is for the recompression designs. Due to its proximity to the critical point, the low-temperature simple design requires the most subdivisions to accurately capture the changing properties of carbon dioxide. This requirement is shown in Figure 6.19, which is a plot of the convergence of thermal efficiency for various representative calls to the `optimal_off_design` subroutine. Of the four designs of interest discussed in Sections 6.1 and 6.2, the low-temperature simple design (shown in blue) requires the highest number of sub-heat exchangers to show convergence. Despite requiring more sub-heat exchangers, the designs based on the simple cycle configuration are computationally much faster than the recompression designs, as shown in Figure 6.20.

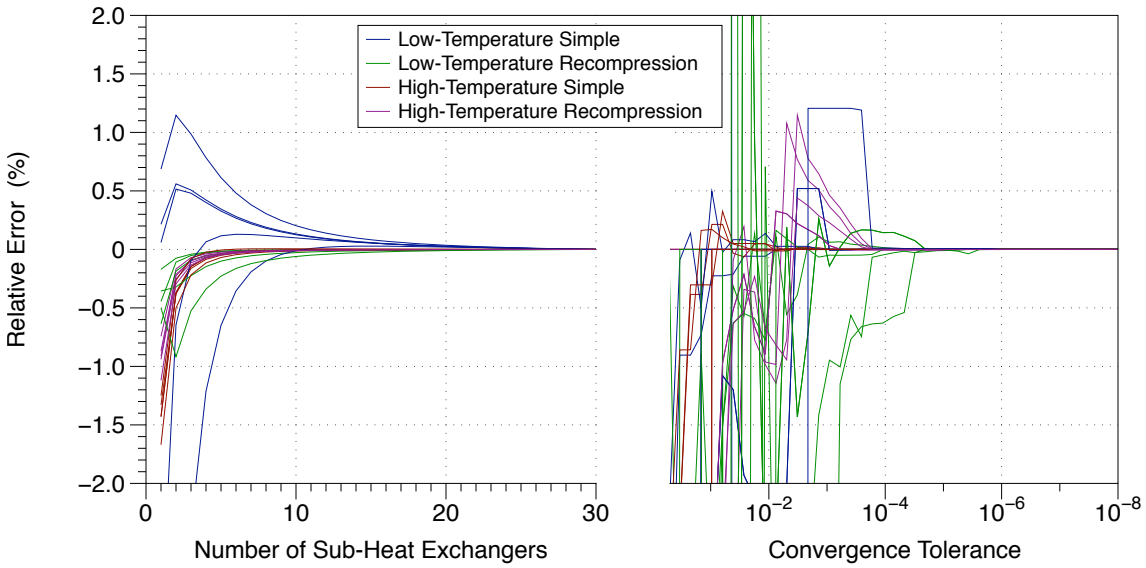


Figure 6.19. Convergence of the `optimal_off_design` subroutine.

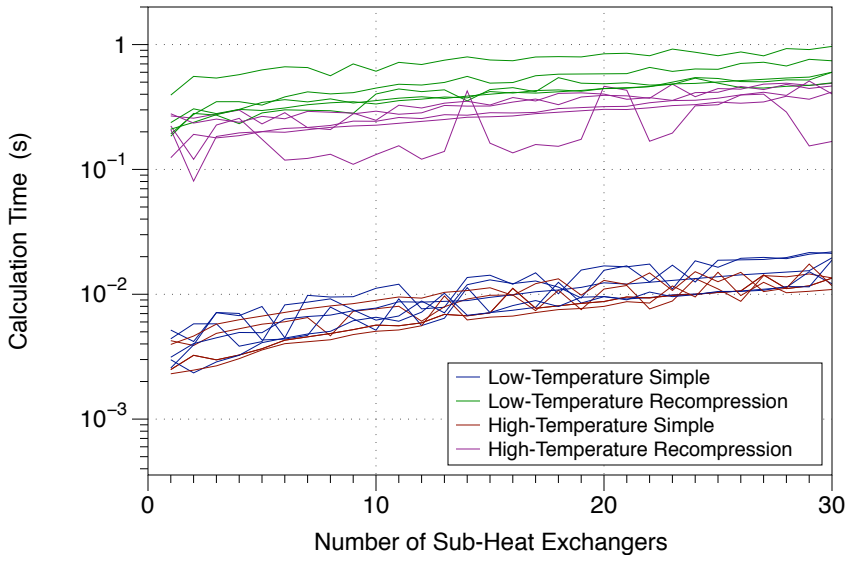


Figure 6.20. Model runtime for the four designs, using the `optimal_off_design` subroutine with compressor inlet pressure specified for a number of representative conditions.

For calls to the `optimal_off_design` with the low-side pressure specified, the simple designs are an order of magnitude faster because the only free parameter is shaft speed, while the recompression designs are optimizing on shaft speed and recompression fraction. The plotted data were generated on a 2.6 GHz Intel Core i7 using the gfortran (version 4.9) compiler, and the variations in the calculation time are due to numerical noise and slightly different starting positions while optimizing thermal efficiency with the subplex library.

7. Conclusions

Supercritical carbon dioxide (SCO_2) power cycles show promise for utility scale electricity production and are generating interest in various industries, including nuclear and concentrating solar power (CSP). SCO_2 power cycles present a number of interesting opportunities for study over a range of disciplines such as turbomachinery design, SCO_2 fluid flow and heat transfer, and corrosion resistance, as well as system-level questions such as the impact of dry cooling and potential control strategies. As such, there is need for high-level analysis of the design-point thermodynamic potential and off-design operating characteristics of these systems in order to provide insight on the most beneficial directions for these investigations. Furthermore, the study of specific SCO_2 applications (e.g., a dry-cooled CSP plant with thermal storage in a particular location) will require detailed models that take into account the specific design under consideration. These detailed models will require a core cycle modeling framework that is computationally efficient and flexible enough to accommodate various design parameters or specific components.

The modeling framework developed for this work effort is intended to satisfy both of these needs. Namely, the developed models are able to provide high-level observations into the design-point and off-design performance of the simple and recompression SCO_2 cycle configurations and are designed with computational efficiency and flexibility in mind. The modeling framework and examples of its usage are freely available online[3] and are intended as building blocks to enable future investigations.

7.1. Design-Point Considerations

The design-point performance of a SCO_2 recompression cycle depends on a number of factors, including its pressure ratio, the proximity of the compressor inlet conditions to the pseudocritical region of

3. http://sel.me.wisc.edu/software.shtml

carbon dioxide, and the trade-off between recompressor work and a more balanced low-temperature recuperator. The analyses discussed in Chapter 5 show that there is no "best" design. Rather, the optimal balance of these factors depends on a number of design decisions, and ultimately the relative merit of various designs and design points will depend on the intended application and component costs. One interesting conclusion from the results presented in Chapter 5 is that increasing the design high-side pressure does not always correspond to higher overall cycle thermal efficiency. Rather, there is an optimal compressor outlet pressure that is dependent on the operating temperatures of the cycle and is typically in the range of 30 to 35 MPa when the recuperators reach a minimum threshold size.

Results also indicate that the recompression cycle is not inherently more efficient than the simple recuperated cycle. For cycles with smaller recuperators, the penalty associated with the imbalance in the heat exchanger does not justify the additional work required by the recompressing compressor. In general, operating at warmer heat rejection temperatures requires increased compressor inlet pressures to maximize cycle efficiency. Warmer heat rejection temperatures also reduce the required recompression fraction of the cycle because it is operating further from the critical temperature of carbon dioxide, resulting in a more naturally-balanced heat transfer process recuperation.

7.2. Off-Design Considerations

The off-design performance characteristics of the designs considered suggest that, for applications where off-design operation is a possibility, it may be better to design for warmer compressor inlet temperatures. Designing for colder compressor inlet temperatures will result in higher design-point efficiencies, but operating these designs under warmer off-design conditions results in limited power output and a reduced operating envelope. The degradation in off-design thermal efficiency as the compressor inlet temperature increases is largely driven by the increase in the turbine's ratio of tip speed to spouting velocity, which reduces turbine efficiency. For a low-reaction radial turbine, the off-design efficiency of the turbine decreases significantly as this ratio approaches unity. For a turbine with a non-negligible reaction ratio, such as the turbine being investigated at SNL, the off-design efficiency of the turbine does not in-

crease as significantly at larger ratios of tip speed to spouting velocity. The observation that low-temperature designs have a more limited off-design operating envelope is significant, as it suggests that investigations into maximizing the design-point efficiency of SCO_2 power cycles under conditions very near the critical point of carbon dioxide are mainly advantageous to power plants designed to rarely operate under off-design low-side temperatures. However, SCO_2 power plants expected to operate over a large range of low-side temperatures, as is likely for dry-cooled designs, may benefit from a warmer design point. The optimal design point is application specific, and the core cycle models developed here are intended to be used for these types of investigations.

The off-design analyses presented in Chapter 6 suggest that inventory control is very beneficial for maximizing off-design thermal efficiency at constant power output. In general, increasing the off-design compressor inlet temperature requires an increase in the low-side pressure of the cycle, though the high-pressure limit of the equipment must be considered. The off-design operation of the simple and recompression cycles are similar, and both designs have limited operational range when designed near the critical point. Away from the critical point, the efficiency advantage of the recompression cycle is less substantial; whether the additional heat exchanger and compressor required by that configuration are justified will depend on the specific application being considered.

7.3. Recommendations for Further Study

The developed modeling framework enables a number of interesting studies. For example, the ability of the cycle to take advantage of off-design compressor inlet conditions below the critical temperature is not clear. The results in Chapter 6 suggest that decreasing the low-side temperature will require a corresponding decrease in low-side pressure, but temperatures and pressures below the critical point could result in two-phase conditions. One possibility to prevent two-phase conditions from occurring (assuming the system is not designed to handle trans-critical operation) is to limit the low-side pressure to values above the critical pressure, but this may result in the cycle's turbomachinery operating outside its valid range. Another option is to introduce additional bypass controls into the cycle in order to prevent the out-

put of the cycle from exceeding its rated power, though the effect of such a bypass on the overall efficiency of the cycle must be considered.

Using a two-stage recompressor expanded the operational envelopes of the considered designs. It is not clear if using a three-stage recompressor or a two-stage main compressor would further expand the range of possible operating conditions for the cycles. The effect on the off-design operating characteristics of switching to an axial turbine is also of interest. The flexible module-based nature of the code facilitates these types of analyses. For example, any model of an axial turbine that satisfies the interface requirements defined in Chapters 3 and 4 can be linked to the cycle models at compile time, and the parametric studies presented in Chapter 6 can easily be repeated with little to no code modification required. This capability allows various types and designs of turbomachinery to be considered with minimal effort.

The models can be used to investigate seasonal or annual performance of a specific power plant; these types of analyses will provide insight into the long-term optimal design-point of a system. The annual performance of a system will need to take into account additional losses such as the power requirements of the heat rejection system. An advantage of the core cycle models is that they do not assume any specific type of heat rejection system. This approach allows the models to be coupled to any system by using the required heat rate in the precooler that is predicted by the cycle model as an input into a model of the heat rejection system. Likewise, the required rate of heat into the cycle at the primary heat exchanger can be used to couple the cycle to any applicable heat source. For example, modeling a CSP plant without storage capabilities requires the rate of heat input to the cycle to be specified based on the available solar irradiance. More complicated interactions (e.g., if the CO_2 temperature entering the precooler affects the amount of heat it can reject) may require iteration between the models, but the computational efficiency of the core cycle models enables this approach. In general, the modeling framework and core cycle models presented in this dissertation are intended as tools to be used for more specific simulations involving SCO_2 power cycles.

References

Angelino, G., "Carbon dioxide condensation cycles for power production," *Journal of Engineering for Power*, Vol. 90, pp. 287, (1968).

Angelino, G., "Real Gas Effects In Carbon Dioxide Cycles", ASME Paper 69-GT-103, (1969).

Brainerd, W. S., *Guide to Fortran 2003 Programming,* Springer, (2009).

Brent, R. P., *Algorithms for Minimization without Derivatives,* Courier Dover Publications, (1973).

Browne, S., Dongarra, J., Grosse, E., & Rowan, T., "The Netlib mathematical software repository," *D-Lib Magazine*, access online: http://www.netlib.org, (1995).

Bryant, J. C., Saari, H., & Zanganeh, K., "An Analysis and Comparison of the Simple and Recompression Supercritical CO_2 Cycles", Proceedings from the Supercritical CO_2 Power Cycle Symposium, Boulder, CO, (2011).

Carstens, N. A., *Control Strategies for Supercritical Carbon Dioxide Power Conversion Systems*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, (2007).

Carter, N. T., & Campbell, R. J., "Water Issues of Concentrating Solar Power (CSP) Electricity in the U.S. Southwest", CRS Report for Congress, No. 7-5700, (2009).

Chacartegui, R., Muñoz de Escalona, J. M., Sánchez, D., Monje, B., & Sánchez, T., "Alternative cycles based on carbon dioxide for central receiver solar power plants," *Applied Thermal Engineering*, Vol. 31, No. 5, pp. 872-879, (2011).

Chapra, S., & Canale, R., *Numerical Methods for Engineers, Sixth Edition,* 6th ed., McGraw-Hill Science/Engineering/Math, (2009).

Chen, H., & Baines, N. C., "The aerodynamic loading of radial and mixed-flow turbines," *International Journal of Mechanical Sciences*, Vol. 36, No. 1, pp. 63-79, (1994).

Conboy, T., Wright, S., Pasch, J., Fleming, D., Rochau, G., & Fuller, R., "Performance Characteristics of an Operating Supercritical CO_2 Brayton Cycle," *J. Eng. Gas Turb. Power*, Vol. 134, No. 11, pp. 111703, (2012).

DOE (U.S. Department of Energy), "Energy Demands on Water Resources", Report to Congress, (2006).

Dostal, V., Hejzlar, P., & Driscoll, M. J., "High-performance supercritical carbon dioxide cycle for next-generation nuclear reactors," *Nucl. Technol.*, Vol. 154, No. 3, pp. 265-282, (2006).

Dostal, V., *A Supercritical Carbon Dioxide Cycle for Next Generation Nuclear Reactors*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, (2004).

F-Chart Software, *Engineering Equation Solver (EES)*, access online: http://www.fchart.com/ees, (2013).

Feher, E. G., "The Supercritical Thermodynamic Power Cycle", Proceedings from Intersociety Energy Conversion Engineering Conference, (1967).

Fox, R. W., Pritchard, P. J., & McDonald, A. T., *Introduction to Fluid Mechanics,* 7th ed., Wiley, (2008).

Gibbs, J. P., Hejzlar, P., & Driscoll, M. J., "Applicability of Supercritical CO_2 Power Conversion Systems to GEN IV Reactors", MIT, Topical Report No. MIT-GFR-037, (2006).

Hexemer, M. J., & Rahner, K., "Supercritical CO_2 Brayton Cycle Integrated System Test (IST) TRACE Model and Control System Design", Proceedings from the Supercritical CO_2 Power Cycle Symposium, Boulder, CO, (2011).

Ishiyama, S., Muto, Y., Kato, Y., Nishio, S., Hayashi, T., & Nomoto, Y., "Study of steam, helium and supercritical CO_2 turbine power generations in prototype fusion power reactor," *Progress in Nuclear Energy*, Vol. 50, No. 2-6, pp. 325-332, (2008).

Japikse, D., & Baines, N. C., *Introduction to Turbomachinery,* Concepts ETI, (1997).

Kenny, J. F., Barber, N. L., Hutson, S. S., Linsey, K. S., Lovelace, J. K., & Maupin, M. A., "Estimated Use of Water in the United States in 2005", U.S. Geological Survey Circular 1344, (2009).

Legault, D. M., *Development and Application of a Steady State Code for Supercritical Carbon Dioxide Cycles*, BS Thesis, Massachusetts Institute of Technology, Cambridge, MA, (2006).

Lemmon, E. W., Huber, M. L., & McLinden, M. O., *NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP*, http://www.nist.gov/srd/nist23.cfm, (2013).

Ludington, A. R., *Tools for Supercritical Carbon Dioxide Cycle Analysis and the Cycle's Applicability to Sodium Fast Reactors*, MS Thesis, Massachusetts Institute of Technology, Cambridge, MA, (2009).

Moisseytsev, A., & Sienicki, J. J., "Investigation of alternative layouts for the supercritical carbon dioxide Brayton cycle for a sodium-cooled fast reactor," *Nuclear Engineering and Design*, Vol. 239, No. 7, pp. 1362-1371, (2009).

Moisseytsev, A., & Sienicki, J. J., "Validation of the ANL Plant Dynamics Code Compressor Model with SNL/BNI Compressor Test Data", Proceedings from Supercritical CO_2 Power Cycle Symposium, Boulder, CO, (2011).

Neises, T., & Turchi, C., "A comparison of supercritical carbon dioxide power cycle configurations with an emphasis on CSP applications", Proceedings from SolarPACES 2013, Las Vegas, NV, (2013).

Nellis, G., & Klein, S., *Heat Transfer,* Cambridge University Press, (2012).

Noall, J., & Forsha, M., "Off-Design Turbomachinery Performance Mapping", Report Prepared by Barber-Nichols, (2008).

Northland Numerics, access online: http://www.northlandnumerics.com, (2014).

Patnode, A. M., *Simulation and Performance Evaluation of Parabolic Trough Solar Power Plants*, MS Thesis, University of Wisconsin-Madison, Madison, WI, (2006).

Peng, W. W., *Fundamentals of Turbomachinery,* 1st ed., Hoboken, NJ, John Wiley & Sons, (2008).

Rowan, T., *Functional Stability Analysis of Numerical Algorithms*, Ph.D. Thesis, University of Texas at Austin, Austin, TX, (1990).

Sarkar, J., & Bhattacharyya, S., "Optimization of recompression S-CO_2 power cycle with reheating," *Energy Conversion and Management*, Vol. 50, No. 8, pp. 1939-1945, (2009).

Sarkar, J., "Second law analysis of supercritical CO_2 recompression Brayton cycle," *Energy*, Vol. 34, No. 9, pp. 1172-1178, (2009).

Span, R., & Wagner, W., "A New Equation of State for Carbon Dioxide Covering the Fluid Region from the Triple-Point Temperature to 1100 K at Pressures up to 800 MPa," *J. Phys. Chem. Ref. Data*, Vol. 25, No. 6, pp. 1509, (1996).

Utamura, M., "Thermodynamic Analysis of Part-Flow Cycle Supercritical CO_2 Gas Turbines," *J. Eng. Gas Turb. Power*, Vol. 132, No. 11, pp. 111701, (2010).

Wright, S. A., Conboy, T. M., Parma, E. J., Lewis, T. G., Rochau, G. A., & Suo-Anttila, A. J., "Summary of the Sandia Supercritical CO_2 Development Program", Proceedings from the Supercritical CO_2 Power Cycle Symposium, Boulder, CO, (2011).

Wright, S. A., Radel, R. F., Vernon, M. E., Rochau, G. E., & Pickard, P. S., "Operation and Analysis of a Supercritical CO_2 Brayton Cycle", Sandia Report SAND2010-0171, (2010).

Appendix I: core Module

core.f90

```
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This file contains the the module 'core', which holds a number of user-defined types, functions, and subroutines required by the
! other SCO2 Brayton cycle modeling modules.  This module also exposes the integer parameter 'dp', which should be used when
! declaring or setting double precision values (e.g., real(dp) :: value = 42.0_dp).
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: August 15, 2014
!
!-----------------------------------------------------------------------------------------------------------------------------------

module core

implicit none
integer, parameter :: dp = selected_real_kind(15)  ! corresponds to double precision

type Compressor
    real(dp) :: D_rotor = 0.0_dp      ! rotor diameter (m)
    real(dp) :: D_rotor_2 = 0.0_dp    ! secondary rotor diameter (m) [used for two-stage recompressor, if necessary]
    real(dp) :: N_design = 0.0_dp     ! design-point shaft speed (rpm)
    real(dp) :: eta_design = 0.0_dp   ! design-point isentropic efficiency (-) [or stage efficiency in two-stage recompressor]
    real(dp) :: phi_design = 0.0_dp   ! design-point flow coefficient (-)
    real(dp) :: phi_min = 0.0_dp      ! surge limit (-)
    real(dp) :: phi_max = 0.0_dp      ! choke limit / zero pressure rise limit / x-intercept (-)
    real(dp) :: N = 0.0_dp            ! shaft speed (rpm)
    real(dp) :: eta = 0.0_dp          ! isentropic efficiency (-)
    real(dp) :: phi = 0.0_dp          ! dimensionless flow coefficient (-)
    real(dp) :: phi_2 = 0.0_dp        ! secondary dimensionless flow coefficient (-) [used for second stage phi, if necessary]
    real(dp) :: w_tip_ratio = 0.0_dp  ! ratio of the local (comp outlet) speed of sound to the tip speed (-)
    logical  :: surge = .false.       ! true if the compressor is in the surge region
end type Compressor

type Turbine
    real(dp) :: D_rotor = 0.0_dp      ! rotor diameter (m)
    real(dp) :: A_nozzle = 0.0_dp     ! effective nozzle area (m2)
    real(dp) :: N_design = 0.0_dp     ! design-point shaft speed (rpm)
    real(dp) :: eta_design = 0.0_dp   ! design-point isentropic efficiency (-)
    real(dp) :: N = 0.0_dp            ! shaft speed (rpm)
    real(dp) :: eta = 0.0_dp          ! isentropic efficiency (-)
    real(dp) :: nu = 0.0_dp           ! ratio of tip speed to spouting velocity (-)
    real(dp) :: w_tip_ratio = 0.0_dp  ! ratio of the local (turbine inlet) speed of sound to the tip speed (-)
end type Turbine

type HeatExchanger
    ! Under design conditions, streams are defined as cold (1) and hot (2).
    real(dp) :: UA_design = 0.0_dp                  ! design-point conductance (kW/K)
    real(dp), dimension(2) :: DP_design = 0.0_dp    ! design-point pressure drops across the heat exchanger (kPa)
    real(dp), dimension(2) :: m_dot_design = 0.0_dp  ! design-point mass flow rates of the two streams (kg/s)
    real(dp) :: Q_dot = 0.0_dp                      ! heat transfer rate (kW)
    real(dp) :: UA = 0.0_dp                         ! conductance (kW/K)
    real(dp) :: min_DT = 0.0_dp                     ! minimum temperature difference in hxr (K)
    real(dp) :: eff = 0.0_dp                        ! heat exchanger effectiveness (-)
    real(dp) :: C_dot_cold = 0.0_dp                 ! cold stream capacitance rate (kW/K)
    real(dp) :: C_dot_hot = 0.0_dp                  ! hot stream capacitance rate (kW/K)
    integer  :: N_sub = 1                           ! number of sub-heat exchangers used in model
end type HeatExchanger

type RecompCycle
    real(dp) :: W_dot_net                                   ! net power output of the cycle (kW)
    real(dp) :: eta_thermal                                 ! thermal efficiency of the cycle (-)
    real(dp) :: recomp_frac                                 ! amount of flow that bypasses the precooler and is compressed in ...
    real(dp) :: m_dot_turbine                               ! mass flow rate through the turbine (kg/s)
    real(dp) :: high_pressure_limit                         ! maximum allowable high-side pressure (kPa)
    real(dp) :: conv_tol                                    ! relative convergence tolerance used during iteration loops involving ...
    type(Turbine) :: t                                      ! turbine user-defined type
    type(Compressor) :: mc, rc                              ! compressor and recompressor user-defined types
    type(HeatExchanger) :: LT, HT, PHX, PC                  ! heat exchanger user-defined types
    real(dp), dimension(10) :: temp, pres, enth, entr, dens  ! thermodynamic properties at the state points of the cycle (K, kPa, ...
end type RecompCycle
```

```
type ErrorTrace
    integer :: code = 0                ! the generated error code
    integer, dimension(4) :: lines = 0  ! the lines of the calls that generated the error
    integer, dimension(4) :: files = 0  ! the files of the calls that generated the error, using:
end type ErrorTrace                    !   1: core, 2: design_point, 3: off_design_point, 4: compressors, 5: turbines, 6: heat ...


contains


subroutine calculate_turbomachine_outlet( &
    T_in, P_in, P_out, eta, is_comp, error_trace, enth_in, entr_in, dens_in, temp_out, enth_out, entr_out, dens_out, spec_work &
    )
    ! Determine the outlet state of a compressor or turbine using isentropic efficiency and outlet pressure.
    !
    ! Inputs:
    !   T_in -- inlet temperature (K)
    !   P_in -- inlet pressure (kPa)
    !   P_out -- outlet pressure (kPa)
    !   eta -- isentropic efficiency (-)
    !   is_comp -- if .true., model a compressor (w = w_s / eta); if .false., model a turbine (w = w_s * eta)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   enth_in -- inlet specific enthalpy (kJ/kg) [optional]
    !   entr_in -- inlet specific entropy (kJ/kg-K) [optional]
    !   dens_in -- inlet fluid density (kg/m3) [optional]
    !   temp_out -- outlet fluid temperature (K) [optional]
    !   enth_out -- outlet specific enthalpy (kJ/kg) [optional]
    !   entr_out -- outlet specific entropy (kJ/kg-K) [optional]
    !   dens_out -- outlet fluid density (kg/m3) [optional]
    !   spec_work -- specific work of the turbomachine (kJ/kg) [optional]
    !
    ! Notes:
    !   1) The specific work of the turbomachine is positive for a turbine and negative for a compressor.
    !   2) No error checking is performed on the inlet and outlet pressures; valid pressure ratios are assumed.

    use CO2_Properties, only: CO2_TP, CO2_PS, CO2_PH

    ! Arguments
    real(dp), intent(in) :: T_in, P_in, P_out, eta
    logical, intent(in)  :: is_comp
    type(ErrorTrace), intent(out) :: error_trace
    real(dp), intent(out), optional :: enth_in, entr_in, dens_in, temp_out, enth_out, entr_out, dens_out, spec_work

    ! Local Variables
    real(dp) :: h_in, s_in, h_s_out, w_s, w, h_out
    integer  :: error_code

    call CO2_TP(T=T_in, P=P_in, error_code=error_code, enth=h_in, entr=s_in, dens=dens_in)  ! properties at the inlet conditions
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 134
        error_trace%files(1) = 1
        return
    end if

    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet enthalpy if compression/expansion is isentropic
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 142
        error_trace%files(1) = 1
        return
    end if

    w_s = h_in - h_s_out  ! specific work if process is isentropic (negative for compression, positive for expansion)
    if (is_comp) then
        w = w_s / eta     ! actual specific work of compressor (negative value)
    else
        w = w_s * eta     ! actual specific work of turbine (positive value)
    end if
    h_out = h_in - w      ! energy balance on turbomachine

    call CO2_PH(P=P_out, H=h_out, error_code=error_code, temp=temp_out, entr=entr_out, dens=dens_out)  ! properties at the outlet
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 158
        error_trace%files(1) = 1
        return
    end if

    if (present(enth_in)) enth_in = h_in
    if (present(entr_in)) entr_in = s_in
    if (present(enth_out)) enth_out = h_out
    if (present(spec_work)) spec_work = w

end subroutine calculate_turbomachine_outlet
```

```
subroutine isen_eta_from_poly_eta(T_in, P_in, P_out, poly_eta, is_comp, error_trace, isen_eta)
    ! Calculate the isentropic efficiency that corresponds to a given polytropic efficiency
    ! for the expansion or compression from T_in and P_in to P_out.
    !
    ! Inputs:
    !   T_in -- inlet temperature (K)
    !   P_in -- inlet pressure (kPa)
    !   P_out -- outlet pressure (kPa)
    !   poly_eta -- polytropic efficiency (-)
    !   is_comp -- if .true., model a compressor (w = w_s / eta); if .false., model a turbine (w = w_s * eta)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   isen_eta -- the equivalent isentropic efficiency (-)
    !
    ! Notes:
    !   1) Integration of small DP is approximated numerically by using 200 stages.
    !   2) No error checking is performed on the inlet and outlet pressures; valid pressure ratios are assumed.

    use CO2_Properties, only: CO2_TP, CO2_PS, CO2_PH

    ! Arguments
    real(dp), intent(in)  :: T_in, P_in, P_out, poly_eta
    logical, intent(in)   :: is_comp
    real(dp), intent(out) :: isen_eta
    type(ErrorTrace), intent(out) :: error_trace

    ! Parameters
    integer, parameter :: stages = 200

    ! Local Variables
    real(dp) :: h_in, s_in, h_s_out, w_s, w, stage_DP
    real(dp) :: stage_P_in, stage_P_out, stage_h_in, stage_s_in, stage_h_s_out, stage_h_out
    integer  :: error_code, stage

    call CO2_TP(T=T_in, P=P_in, error_code=error_code, enth=h_in, entr=s_in)  ! properties at the inlet conditions
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 209
        error_trace%files(1) = 1
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet enthalpy if compression/expansion is isentropic
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 216
        error_trace%files(1) = 1
        return
    end if

    stage_P_in = P_in   ! initialize first stage inlet pressure
    stage_h_in = h_in   ! initialize first stage inlet enthalpy
    stage_s_in = s_in   ! initialize first stage inlet entropy
    stage_DP = (P_out - P_in) / real(stages,dp)  ! pressure change per stage
    do stage = 1, stages
        stage_P_out = stage_P_in + stage_DP
        call CO2_PS(P=stage_P_out, S=stage_s_in, error_code=error_code, enth=stage_h_s_out)  ! outlet enthalpy if compression/expansion ...
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 230
            error_trace%files(1) = 1
            return
        end if
        w_s = stage_h_in - stage_h_s_out  ! specific work if process is isentropic
        if (is_comp) then
            w = w_s / poly_eta            ! actual specific work of compressor (negative value)
        else
            w = w_s * poly_eta            ! actual specific work of turbine (positive value)
        end if
        stage_h_out = stage_h_in - w      ! energy balance on stage

        ! Reset next stage inlet values.
        stage_P_in = stage_P_out
        stage_h_in = stage_h_out
        call CO2_PH(P=stage_P_in, H=stage_h_in, error_code=error_code, entr=stage_s_in)
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 248
            error_trace%files(1) = 1
            return
        end if

    end do

    ! Note: last stage outlet enthalpy is equivalent to turbomachine outlet enthalpy.
    if (is_comp) then
        isen_eta = (h_s_out - h_in) / (stage_h_out - h_in)
    else
        isen_eta = (stage_h_out - h_in) / (h_s_out - h_in)
    end if

end subroutine isen_eta_from_poly_eta
```

```
subroutine calculate_hxr_UA( &
    N_sub_hxrs, Q_dot, m_dot_c, m_dot_h, T_c_in, T_h_in, P_c_in, P_c_out, P_h_in, P_h_out, error_trace, UA, min_DT &
    )
    ! Calculate the conductance (UA value) and minimum temperature difference of a heat exchanger
    ! given its mass flow rates, inlet temperatures, and a rate of heat transfer.
    !
    ! Inputs:
    !   N_sub_hxrs -- the number of sub-heat exchangers to use for discretization
    !   Q_dot -- rate of heat transfer in the heat exchanger (kW)
    !   m_dot_c -- cold stream mass flow rate (kg/s)
    !   m_dot_h -- hot stream mass flow rate (kg/s)
    !   T_c_in -- cold stream inlet temperature (K)
    !   T_h_in -- hot stream inlet temperature (K)
    !   P_c_in -- cold stream inlet pressure (kPa)
    !   P_c_out -- cold stream outlet pressure (kPa)
    !   P_h_in -- hot stream inlet pressure (kPa)
    !   P_h_out -- hot stream outlet pressure (kPa)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   UA -- heat exchanger conductance (kW/K)
    !   min_DT -- minimum temperature difference ("pinch point") between hot and cold streams in heat exchanger (K)
    !
    ! Notes:
    !   1) Total pressure drop for each stream is divided equally among the sub-heat exchangers (i.e., DP is a linear distribution).

    use CO2_Properties, only: CO2_TP, CO2_PH

    ! Arguments
    integer, intent(in)   :: N_sub_hxrs
    real(dp), intent(in)  :: Q_dot, m_dot_c, m_dot_h, T_c_in, T_h_in, P_c_in, P_c_out, P_h_in, P_h_out
    real(dp), intent(out) :: UA, min_DT
    type(ErrorTrace), intent(out) :: error_trace

    ! Local Variables
    integer  :: i, error_code
    real(dp) :: h_c_in, h_h_in, h_c_out, h_h_out
    real(dp), dimension(N_sub_hxrs+1) :: P_c, P_h, T_c, T_h, h_c, h_h
    real(dp), dimension(N_sub_hxrs) :: C_dot_c, C_dot_h, C_dot_min, C_dot_max, C_R, eff, NTU

    ! Check inputs.
    if (T_h_in < T_c_in) then
        error_trace%code = 5
        error_trace%lines(1) = 309
        error_trace%files(1) = 1
        return
    end if
    if (P_h_in < P_h_out) then
        error_trace%code = 6
        error_trace%lines(1) = 315
        error_trace%files(1) = 1
        return
    end if
    if (P_c_in < P_c_out) then
        error_trace%code = 7
        error_trace%lines(1) = 321
        error_trace%files(1) = 1
        return
    end if
    if (abs(Q_dot) <= 1d-12) then  ! very low Q_dot; assume it is zero
        UA = 0.0_dp
        min_DT = T_h_in - T_c_in
        return
    end if

    ! Assume pressure varies linearly through heat exchanger.
    P_c = [ ( P_c_out + i * (P_c_in - P_c_out) / real(N_sub_hxrs,dp) , i = 0, N_sub_hxrs ) ]  ! create linear vector of cold stream ...
    P_h = [ ( P_h_in - i * (P_h_in - P_h_out) / real(N_sub_hxrs,dp) , i = 0, N_sub_hxrs ) ]   ! create linear vector of hot stream ...

    ! Calculate inlet enthalpies from known state points.
    call CO2_TP(T=T_c_in, P=P_c(N_sub_hxrs+1), error_code=error_code, enth=h_c_in)
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 338
        error_trace%files(1) = 1
        return
    end if
    call CO2_TP(T=T_h_in, P=P_h(1), error_code=error_code, enth=h_h_in)
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 345
        error_trace%files(1) = 1
        return
    end if

    ! Calculate outlet enthalpies from energy balances.
    h_c_out = h_c_in + Q_dot / m_dot_c
    h_h_out = h_h_in - Q_dot / m_dot_h
```

```
    ! Set up the enthalpy vectors and loop through the sub-heat exchangers, calculating temperatures.
    h_c = [ ( h_c_out + i * (h_c_in - h_c_out) / real(N_sub_hxrs,dp) , i = 0, N_sub_hxrs ) ]  ! create linear vector of cold ...
    h_h = [ ( h_h_in - i * (h_h_in - h_h_out) / real(N_sub_hxrs,dp) , i = 0, N_sub_hxrs ) ]   ! create linear vector of hot ...
    T_h(1) = T_h_in  ! hot stream inlet temperature
    call CO2_PH(P=P_c(1), H=h_c(1), error_code=error_code, temp=T_c(1))  ! cold stream outlet temperature
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 361
        error_trace%files(1) = 1
        return
    end if
    if (T_c(1) >= T_h(1)) then  ! there was a second law violation in this sub-heat exchanger
        error_trace%code = 11
        error_trace%lines(1) = 368
        error_trace%files(1) = 1
        return
    end if
    do i = 2,N_sub_hxrs+1
        call CO2_PH(P=P_h(i), H=h_h(i), error_code=error_code, temp=T_h(i))
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 375
            error_trace%files(1) = 1
            return
        end if
        call CO2_PH(P=P_c(i), H=h_c(i), error_code=error_code, temp=T_c(i))
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 382
            error_trace%files(1) = 1
            return
        end if
        if (T_c(i) >= T_h(i)) then  ! there was a second law violation in this sub-heat exchanger
            error_trace%code = 11
            error_trace%lines(1) = 389
            error_trace%files(1) = 1
            return
        end if
    end do

    ! Perform effectiveness-NTU and UA calculations (note: the below are all array operations).
    C_dot_h = m_dot_h * (h_h(1:N_sub_hxrs) - h_h(2:N_sub_hxrs+1)) / (T_h(1:N_sub_hxrs) - T_h(2:N_sub_hxrs+1))  ! hot stream capacitance rate
    C_dot_c = m_dot_c * (h_c(1:N_sub_hxrs) - h_c(2:N_sub_hxrs+1)) / (T_c(1:N_sub_hxrs) - T_c(2:N_sub_hxrs+1))  ! cold stream capacitance rate
    C_dot_min = min(C_dot_h, C_dot_c)  ! minimum capacitance stream
    C_dot_max = max(C_dot_h, C_dot_c)  ! maximum capacitance stream
    C_R = C_dot_min / C_dot_max        ! capacitance ratio of sub-heat exchanger
    eff = Q_dot / ((N_sub_hxrs * C_dot_min * (T_h(1:N_sub_hxrs) - T_c(2:N_sub_hxrs+1))))  ! effectiveness of each sub-heat exchanger
    where (C_R /= 1.0_dp)
        NTU = log((1.0_dp - eff * C_R) / (1.0_dp - eff)) / (1.0_dp - C_R)  ! NTU if C_R does not equal 1
    elsewhere
        NTU = eff / (1.0_dp - eff)  ! NTU if C_R equals 1
    end where
    UA = sum(NTU * C_dot_min)   ! calculate total UA value for the heat exchanger
    min_DT = minval(T_h - T_c)  ! find the smallest temperature difference within the heat exchanger

    ! Check for NaNs.
    if (UA /= UA) then
        error_trace%code = 14
        error_trace%lines(1) = 413
        error_trace%files(1) = 1
        return
    end if

end subroutine calculate_hxr_UA


integer function next_trace_index(error_trace)
    ! Return the next index that should be used for tracing the lines / files
    ! that generated an error.
    type(ErrorTrace), intent(in) :: error_trace
    next_trace_index = minloc(error_trace%lines, 1)  ! assumes no line numbers are negative; returns location of first 0 in array
end function next_trace_index


end module core
```

Appendix II: design_point Module

design_point.f90

```
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This file contains the the module 'design_point', which defines three system-level subroutines:
!   design -- the main design-point model
!   optimal_design -- calls 'design' and incorporates optimization in order to maximize thermal efficiency by varying model inputs
!   auto_optimal_design -- calls 'optimal_design' with multiple starting points in an effort to find the global maximum for thermal
!                          efficiency, but is significantly slower (if you have a good idea what the design point should be, it is
!                          better to use 'optimal_design' with appropriate initial guesses for the inputs)
!
! Notes:
!   1) W_dot_net must be positive.
!   2) Pressure drops are specified per heat exchanger, with stream 1 being the cold stream and stream 2 being the hot stream.
!      Positive values are absolute pressure drops and negative values are relative pressure drops: abs(rel_DP) * P_in = DP.
!   3) Positive values for turbomachinery efficiencies are treated as isentropic, while negative values are treated as polytropic
!      efficiencies (after taking the absolute value).  Using polytropic efficiencies is significantly slower than isentropic.
!
! Cycle State Points:
!   1)  mc in           / PC out
!   2)  LT in (cold)    / mc out
!   3)  mixing valve in / LT out (cold)
!   4)  HT in (cold)    / mixing valve out
!   5)  PHX in          / HT out (cold)
!   6)  turbine in      / PHX out
!   7)  HT in (hot)     / turbine out
!   8)  LT in (hot)     / HT out (hot)
!   9)  PC and rc in    / LT out (hot)
!   10) mixing valve in / recomp out
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: August 15, 2014
!
!-----------------------------------------------------------------------------------------------------------------------------------

module design_point

use core
implicit none
private
public :: design, optimal_design, auto_optimal_design

contains

subroutine design( &
    W_dot_net,     &  ! [input] target net cycle power (kW)
    T_mc_in,       &  ! [input] compressor inlet temperature (K)
    T_t_in,        &  ! [input] turbine inlet temperature (K)
    P_mc_in,       &  ! [input] compressor inlet pressure (kPa)
    P_mc_out,      &  ! [input] compressor outlet pressure (kPa)
    DP_LT,         &  ! [input] pressure drops in low-temperature recuperator (kPa if positive values)
    DP_HT,         &  ! [input] pressure drops in high-temperature recuperator (kPa if positive values)
    DP_PC,         &  ! [input] pressure drops in precooler (kPa if positive values)
    DP_PHX,        &  ! [input] pressure drops in primary heat exchanger (kPa if positive values)
    UA_LT,         &  ! [input] design-point UA value for the low-temperature recuperator (kW/K)
    UA_HT,         &  ! [input] design-point UA value for the high-temperature recuperator (kW/K)
    recomp_frac,   &  ! [input] fraction of flow that bypasses the precooler and main compressor at the design point
    eta_mc,        &  ! [input] design-point efficiency of the main compressor; isentropic if positive, polytropic if negative
    eta_rc,        &  ! [input] design-point efficiency of the recompressor; isentropic if positive, polytropic if negative
    eta_t,         &  ! [input] design-point efficiency of the turbine; isentropic if positive, polytropic if negative
    N_sub_hxrs,    &  ! [input] number of sub-heat exchangers to use when calculating UA value for a heat exchanger
    tol,           &  ! [input] convergence tolerance
    error_trace,   &  ! [output] an ErrorTrace object
    recomp_cycle   &  ! [output] a RecompCycle object
    )

    use CO2_Properties, only: CO2_TP, CO2_PH

    ! Arguments
    real(dp), intent(in) :: W_dot_net, T_mc_in, T_t_in, P_mc_in, P_mc_out, UA_LT, UA_HT, recomp_frac
    real(dp), intent(in) :: eta_mc, eta_rc, eta_t, tol
```

```
integer, intent(in)  :: N_sub_hxrs
real(dp), dimension(2), intent(in) :: DP_LT, DP_HT, DP_PC, DP_PHX
type(ErrorTrace), intent(out)  :: error_trace
type(RecompCycle), intent(out) :: recomp_cycle

! Parameters
integer, parameter :: max_iter = 500
real(dp), parameter :: temperature_tolerance = 1.0e-6_dp  ! temperature differences below this are considered zero

! Local Variables
integer  :: T9_iter, T8_iter, error_code, index
real(dp) :: w_mc, w_rc, w_t, C_dot_min, Q_dot_max
real(dp) :: T9_lower_bound, T9_upper_bound, T8_lower_bound, T8_upper_bound, last_LT_residual, last_T9_guess
real(dp) :: last_HT_residual, last_T8_guess, secant_guess
real(dp) :: m_dot_t, m_dot_mc, m_dot_rc, eta_mc_isen, eta_rc_isen, eta_t_isen
real(dp) :: min_DT_LT, min_DT_HT, UA_LT_calc, UA_HT_calc, Q_dot_LT, Q_dot_HT, UA_HT_residual, UA_LT_residual
real(dp), dimension(10) :: temp, pres, enth, entr, dens

! Initialize a few variables.
m_dot_t = 0.0_dp
m_dot_mc = 0.0_dp
m_dot_rc = 0.0_dp
Q_dot_LT = 0.0_dp
Q_dot_HT = 0.0_dp
UA_LT_calc = 0.0_dp
UA_HT_calc = 0.0_dp
temp(1) = T_mc_in
pres(1) = P_mc_in
pres(2) = P_mc_out
temp(6) = T_t_in

! Apply pressure drops to heat exchangers, fully defining the pressures at all states.
if (DP_LT(1) < 0.0_dp) then
    pres(3) = pres(2) - pres(2) * abs(DP_LT(1))   ! relative pressure drop specified for LT recuperator (cold stream)
else
    pres(3) = pres(2) - DP_LT(1)                  ! absolute pressure drop specified for LT recuperator (cold stream)
end if
if (UA_LT < 1.0e-12_dp) pres(3) = pres(2)         ! if there is no LT recuperator, there is no pressure drop
pres(4) = pres(3)                                 ! assume no pressure drop in mixing valve
pres(10) = pres(3)                                ! assume no pressure drop in mixing valve
if (DP_HT(1) < 0.0_dp) then
    pres(5) = pres(4) - pres(4) * abs(DP_HT(1))   ! relative pressure drop specified for HT recuperator (cold stream)
else
    pres(5) = pres(4) - DP_HT(1)                  ! absolute pressure drop specified for HT recuperator (cold stream)
end if
if (UA_HT < 1.0e-12_dp) pres(5) = pres(4)         ! if there is no HT recuperator, there is no pressure drop
if (DP_PHX(1) < 0.0_dp) then
    pres(6) = pres(5) - pres(5) * abs(DP_PHX(1))  ! relative pressure drop specified for PHX
else
    pres(6) = pres(5) - DP_PHX(1)                 ! absolute pressure drop specified for PHX
end if
if (DP_PC(2) < 0.0_dp) then
    pres(9) = pres(1) / (1.0_dp - abs(DP_PC(2)))  ! relative pressure drop specified for precooler: P1=P9-P9*rel_DP => P1=P9*(1-rel_DP)
else
    pres(9) = pres(1) + DP_PC(2)                  ! absolute pressure drop specified for precooler
end if
if (DP_LT(2) < 0.0_dp) then
    pres(8) = pres(9) / (1.0_dp - abs(DP_LT(2)))  ! relative pressure drop specified for LT recuperator (hot stream)
else
    pres(8) = pres(9) + DP_LT(2)                  ! absolute pressure drop specified for LT recuperator (hot stream)
end if
if (UA_LT < 1.0e-12_dp) pres(8) = pres(9)         ! if there is no LT recuperator, there is no pressure drop
if (DP_HT(2) < 0.0_dp) then
    pres(7) = pres(8) / (1.0_dp - abs(DP_HT(2)))  ! relative pressure drop specified for HT recuperator (hot stream)
else
    pres(7) = pres(8) + DP_HT(2)                  ! absolute pressure drop specified for HT recuperator (hot stream)
end if
if (UA_HT < 1.0e-12_dp) pres(7) = pres(8)         ! if there is no HT recuperator, there is no pressure drop

! Determine equivalent isentropic efficiencies for main compressor and turbine, if necessary.
if (eta_mc < 0.0_dp) then
    call isen_eta_from_poly_eta(   &
        T_in = temp(1),            &
        P_in = pres(1),            &
        P_out = pres(2),           &
        poly_eta = abs(eta_mc),    &
        is_comp = .true.,          &
        error_trace = error_trace, &
        isen_eta = eta_mc_isen     &
        )
    if (error_trace%code /= 0) then
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 154
        error_trace%files(index) = 2
        return
    end if
else
    eta_mc_isen = eta_mc
end if
```

```
if (eta_t < 0.0_dp) then
    call isen_eta_from_poly_eta(   &
        T_in = temp(6),            &
        P_in = pres(6),            &
        P_out = pres(7),           &
        poly_eta = abs(eta_t),     &
        is_comp = .false.,         &
        error_trace = error_trace, &
        isen_eta = eta_t_isen      &
        )
    if (error_trace%code /= 0) then
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 173
        error_trace%files(index) = 2
        return
    end if
else
    eta_t_isen = eta_t
end if

! Determine the outlet state and specific work for the main compressor and turbine.
call calculate_turbomachine_outlet( &  ! main compressor
    T_in = temp(1),                 &
    P_in = pres(1),                 &
    P_out = pres(2),                &
    eta = eta_mc_isen,              &
    is_comp = .true.,               &
    error_trace = error_trace,      &
    enth_in = enth(1),              &
    entr_in = entr(1),              &
    dens_in = dens(1),              &
    temp_out = temp(2),             &
    enth_out = enth(2),             &
    entr_out = entr(2),             &
    dens_out = dens(2),             &
    spec_work = w_mc                &
    )
if (error_trace%code /= 0) then
    index = next_trace_index(error_trace)
    error_trace%lines(index) = 193
    error_trace%files(index) = 2
    return
end if
call calculate_turbomachine_outlet( &  ! turbine
    T_in = temp(6),                 &
    P_in = pres(6),                 &
    P_out = pres(7),                &
    eta = eta_t_isen,               &
    is_comp = .false.,              &
    error_trace = error_trace,      &
    enth_in = enth(6),              &
    entr_in = entr(6),              &
    dens_in = dens(6),              &
    temp_out = temp(7),             &
    enth_out = enth(7),             &
    entr_out = entr(7),             &
    dens_out = dens(7),             &
    spec_work = w_t                 &
    )
if (error_trace%code /= 0) then
    index = next_trace_index(error_trace)
    error_trace%lines(index) = 215
    error_trace%files(index) = 2
    return
end if

! Check that this cycle can produce power.
if (recomp_frac >= 1.0d-12) then
    if (eta_rc < 0.0_dp) then  ! need to convert polytropic efficiency to isentropic efficiency
        call isen_eta_from_poly_eta(   &
            T_in = temp(2),            &  ! lowest possible recompressor work occurs when temp(9) == temp(2)
            P_in = pres(9),            &
            P_out = pres(10),          &
            poly_eta = abs(eta_rc),    &
            is_comp = .true.,          &
            error_trace = error_trace, &
            isen_eta = eta_rc_isen     &
            )
        if (error_trace%code /= 0) then
            index = next_trace_index(error_trace)
            error_trace%lines(index) = 241
            error_trace%files(index) = 2
            return
        end if
    else
        eta_rc_isen = eta_rc
    end if
    call calculate_turbomachine_outlet( &
        T_in = temp(2),                 &  ! lowest possible recompressor work occurs when temp(9) == temp(2)
        P_in = pres(9),                 &
        P_out = pres(10),               &
        eta = eta_rc_isen,              &
```

```
        is_comp = .true.,               &
        error_trace = error_trace,      &
        spec_work = w_rc                &
        )
    if (error_trace%code /= 0) then
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 259
        error_trace%files(index) = 2
        return
    end if
else
    w_rc = 0.0_dp
end if
if (w_mc + w_rc + w_t <= 0.0_dp) then  ! positive net power is impossible; return an error
    error_trace%code = 25
    error_trace%lines(1) = 277
    error_trace%files(1) = 2
    return
end if

! Outer iteration loop: temp(8), checking against UA_HT.
if (UA_HT < 1.0e-12_dp) then  ! no high-temperature recuperator
    T8_lower_bound = temp(7)  ! no iteration necessary
    T8_upper_bound = temp(7)  ! no iteration necessary
    temp(8) = temp(7)
    UA_HT_calc = 0.0_dp
    last_HT_residual = 0.0_dp
    last_T8_guess = temp(7)
else
    T8_lower_bound = temp(2)    ! the absolute lowest temp(8) could be
    T8_upper_bound = temp(7)    ! the absolutely highest temp(8) could be
    temp(8) = (T8_lower_bound + T8_upper_bound) * 0.5_dp  ! bisect bounds for first guess
    UA_HT_calc = -1.0_dp
    last_HT_residual = UA_HT    ! know a priori that with T8 = T7, UA_calc = 0 therefore residual is UA_HT - 0.0
    last_T8_guess = temp(7)
end if
T8_loop: do T8_iter = 1,max_iter

    ! Fully define state 8.
    call CO2_TP(T=temp(8), P=pres(8), error_code=error_code, enth=enth(8), entr=entr(8), dens=dens(8))
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 303
        error_trace%files(1) = 2
        return
    end if

    ! Inner iteration loop: temp(9), checking against UA_LT.
    if (UA_LT < 1.0e-12_dp) then  ! no low-temperature recuperator
        T9_lower_bound = temp(8)  ! no iteration necessary
        T9_upper_bound = temp(8)  ! no iteration necessary
        temp(9) = temp(8)
        UA_LT_calc = 0.0_dp
        last_LT_residual = 0.0_dp
        last_T9_guess = temp(8)
    else
        T9_lower_bound = temp(2)    ! the absolute lowest temp(9) could be
        T9_upper_bound = temp(8)    ! the absolutely highest temp(9) could be
        temp(9) = (T9_lower_bound + T9_upper_bound) * 0.5_dp  ! bisect bounds for first guess
        UA_LT_calc = -1.0_dp
        last_LT_residual = UA_LT    ! know a priori that with T9 = T8, UA_calc = 0 therefore residual is UA_LT - 0
        last_T9_guess = temp(8)
    end if
    T9_loop: do T9_iter = 1,max_iter

            ! Determine the outlet state of the recompressing compressor and its specific work.
            if (recomp_frac >= 1.0e-12_dp) then
                if (eta_rc < 0.0_dp) then  ! recalculate isentropic efficiency of recompressing compressor (because T9 changes)
                    call isen_eta_from_poly_eta(   &
                        T_in = temp(9),            &
                        P_in = pres(9),            &
                        P_out = pres(10),          &
                        poly_eta = abs(eta_rc),    &
                        is_comp = .true.,          &
                        error_trace = error_trace, &
                        isen_eta = eta_rc_isen     &
                        )
                    if (error_trace%code /= 0) then
                        index = next_trace_index(error_trace)
                        error_trace%lines(index) = 332
                        error_trace%files(index) = 2
                        return
                    end if
                else
                    eta_rc_isen = eta_rc
                end if
                call calculate_turbomachine_outlet( &
                    T_in = temp(9),                 &
                    P_in = pres(9),                 &
                    P_out = pres(10),               &
                    eta = eta_rc_isen,              &
                    is_comp = .true.,               &
```

```
        error_trace = error_trace,      &
        enth_in = enth(9),              &
        entr_in = entr(9),              &
        dens_in = dens(9),              &
        temp_out = temp(10),            &
        enth_out = enth(10),            &
        entr_out = entr(10),            &
        dens_out = dens(10),            &
        spec_work = w_rc                &
        )
    if (error_trace%code /= 0) then
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 350
        error_trace%files(index) = 2
        return
    end if
else
    w_rc = 0.0_dp  ! the recompressing compressor does not exist
    call CO2_TP(T=temp(9), P=pres(9), error_code=error_code, enth=enth(9), entr=entr(9), dens=dens(9))  ! fully define ...
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 374
        error_trace%files(1) = 2
        return
    end if
    temp(10) = temp(9)  ! assume state 10 is the same as state 9
    enth(10) = enth(9)
    entr(10) = entr(9)
    dens(10) = dens(9)
end if

! Knowing the specific work of the the recompressing compressor, the required mass flow rate can be calculated.
m_dot_t = W_dot_net / (w_mc * (1.0_dp - recomp_frac) + w_rc * recomp_frac + w_t)  ! required mass flow rate through turbine
if (m_dot_t < 0.0_dp) then  ! positive power output is not possible with these inputs
    error_trace%code = 29
    error_trace%lines(1) = 389
    error_trace%files(1) = 2
    return
end if
m_dot_rc = m_dot_t * recomp_frac  ! apply definition of recompression fraction
m_dot_mc = m_dot_t - m_dot_rc     ! mass balance

! Calculate the UA value of the low-temperature recuperator.
if (UA_LT < 1.0e-12_dp) then  ! no low-temp recuperator (this check is necessary to prevent pressure drops with ...
    Q_dot_LT = 0.0_dp
else
    Q_dot_LT = m_dot_t * (enth(8) - enth(9))
end if
call calculate_hxr_UA(         &
    N_sub_hxrs = N_sub_hxrs,   &
    Q_dot = Q_dot_LT,          &
    m_dot_c = m_dot_mc,        &
    m_dot_h = m_dot_t,         &
    T_c_in = temp(2),          &
    T_h_in = temp(8),          &
    P_c_in = pres(2),          &
    P_c_out = pres(3),         &
    P_h_in = pres(8),          &
    P_h_out = pres(9),         &
    error_trace = error_trace, &
    UA = UA_LT_calc,           &
    min_DT = min_DT_LT         &
    )
if (error_trace%code > 0) then
    if (error_trace%code == 11) then  ! second-law violation in hxr, therefore temp(9) is too low
        T9_lower_bound = temp(9)
        temp(9) = (T9_lower_bound + T9_upper_bound) * 0.5_dp  ! bisect bounds for next guess
        error_trace%code = 0  ! reset error trace
        error_trace%lines = 0
        error_trace%files = 0
        cycle T9_loop
    else
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 404
        error_trace%files(index) = 2
        return
    end if
end if

! Check for convergence and adjust T9 appropriately.
UA_LT_residual = UA_LT - UA_LT_calc
if (abs(UA_LT_residual) < 1.0e-12_dp) exit T9_loop  ! catches no LT case
secant_guess = temp(9) - UA_LT_residual * (last_T9_guess - temp(9)) / (last_LT_residual - UA_LT_residual)  ! next guess ...
if (UA_LT_residual < 0.0_dp) then  ! UA_LT_calc is too big, temp(9) needs to be higher
    if (abs(UA_LT_residual)/UA_LT < tol) exit T9_loop  ! UA_LT converged (residual is negative)
    T9_lower_bound = temp(9)
else  ! UA_LT_calc is too small, temp(9) needs to be lower
    if (UA_LT_residual/UA_LT < tol) exit T9_loop  ! UA_LT converged
    if (min_DT_LT < temperature_tolerance) exit T9_loop  ! UA_calc is still too low but there isn't anywhere to go so ...
    T9_upper_bound = temp(9)
end if
last_LT_residual = UA_LT_residual  ! reset last stored residual value
```

```
        last_T9_guess = temp(9)  ! reset last stored guess value

        ! Check if the secant method overshoots and fall back to bisection if it does.
        if (secant_guess <= T9_lower_bound .or. secant_guess >= T9_upper_bound .or. secant_guess /= secant_guess) then  ! secant ...
            temp(9) = (T9_lower_bound + T9_upper_bound) * 0.5_dp
        else
            temp(9) = secant_guess
        end if

end do T9_loop

! Check that T9_loop converged.
if (T9_iter >= max_iter) then
    error_trace%code = 31
    error_trace%lines(1) = 460
    error_trace%files(1) = 2
    return
end if

! State 3 can now be fully defined.
enth(3) = enth(2) + Q_dot_LT / m_dot_mc  ! energy balance on cold stream of low-temp recuperator
call CO2_PH(P=pres(3), H=enth(3), error_code=error_code, temp=temp(3), entr=entr(3), dens=dens(3))
if (error_code /= 0) then
    error_trace%code = error_code
    error_trace%lines(1) = 469
    error_trace%files(1) = 2
    return
end if

! Go through the mixing valve.
if (recomp_frac >= 1.0e-12_dp) then
    enth(4) = (1.0_dp - recomp_frac) * enth(3) + recomp_frac * enth(10)  ! conservation of energy (both sides divided by m_dot_t)
    call CO2_PH(P=pres(4), H=enth(4), error_code=error_code, temp=temp(4), entr=entr(4), dens=dens(4))
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 480
        error_trace%files(1) = 2
        return
    end if
else  ! no mixing valve, therefore state 4 is equal to state 3
    temp(4) = temp(3)
    enth(4) = enth(3)
    entr(4) = entr(3)
    dens(4) = dens(3)
end if

! Check for a second law violation at the outlet of the high-temp recuperator.
if (temp(4) >= temp(8)) then  ! temp(8) is not valid and it must be increased
    T8_lower_bound = temp(8)
    temp(8) = (T8_lower_bound + T8_upper_bound) * 0.5_dp
    cycle T8_loop
end if

! Calculate the UA value of the high-temperature recuperator.
if (UA_HT < 1.0e-12_dp) then  ! no high-temp recuperator
    Q_dot_HT = 0.0_dp
else
    Q_dot_HT = m_dot_t * (enth(7) - enth(8))
end if
call calculate_hxr_UA(         &
    N_sub_hxrs = N_sub_hxrs,   &
    Q_dot = Q_dot_HT,          &
    m_dot_c = m_dot_t,         &
    m_dot_h = m_dot_t,         &
    T_c_in = temp(4),          &
    T_h_in = temp(7),          &
    P_c_in = pres(4),          &
    P_c_out = pres(5),         &
    P_h_in = pres(7),          &
    P_h_out = pres(8),         &
    error_trace = error_trace, &
    UA = UA_HT_calc,           &
    min_DT = min_DT_HT         &
    )
if (error_trace%code > 0) then
    if (error_trace%code == 11) then  ! second-law violation in hxr, therefore temp(8) is too low
        T8_lower_bound = temp(8)
        temp(8) = (T8_lower_bound + T8_upper_bound) * 0.5_dp  ! bisect bounds for next guess
        error_trace%code = 0  ! reset error trace
        error_trace%lines = 0
        error_trace%files = 0
        cycle T8_loop
    else
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 507
        error_trace%files(index) = 2
        return
    end if
end if

! Check for convergence and adjust T8 appropriately.
UA_HT_residual = UA_HT - UA_HT_calc
```

```
        if (abs(UA_HT_residual) < 1.0e-12_dp) exit T8_loop  ! catches no HT case
        secant_guess = temp(8) - UA_HT_residual * (last_T8_guess - temp(8)) / (last_HT_residual - UA_HT_residual)  ! next guess predicted ...
        if (UA_HT_residual < 0.0_dp) then  ! UA_HT_calc is too big, temp(8) needs to be higher
            if (abs(UA_HT_residual)/UA_HT < tol) exit T8_loop  ! UA_HT converged (residual is negative)
            T8_lower_bound = temp(8)
        else  ! UA_HT_calc is too small, temp(8) needs to be lower
            if (UA_HT_residual/UA_HT < tol) exit T8_loop  ! UA_HT converged
            if (min_DT_HT < temperature_tolerance) exit T8_loop  ! UA_calc is still too low but there isn't anywhere to go so it's ok ...
            T8_upper_bound = temp(8)
        end if
        last_HT_residual = UA_HT_residual  ! reset last stored residual value
        last_T8_guess = temp(8)  ! reset last stored guess value

        ! Check if the secant method overshoots and fall back to bisection if it does.
        if (secant_guess <= T8_lower_bound .or. secant_guess >= T8_upper_bound) then  ! secant method overshot, use bisection
            temp(8) = (T8_lower_bound + T8_upper_bound) * 0.5_dp
        else
            temp(8) = secant_guess
        end if

    end do T8_loop

    ! Check that T8_loop converged.
    if (T8_iter >= max_iter) then
        error_trace%code = 35
        error_trace%lines(1) = 563
        error_trace%files(1) = 2
        return
    end if

    ! State 5 can now be fully defined.
    enth(5) = enth(4) + Q_dot_HT / m_dot_t  ! energy balance on cold stream of high-temp recuperator
    call CO2_PH(P=pres(5), H=enth(5), error_code=error_code, temp=temp(5), entr=entr(5), dens=dens(5))
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 572
        error_trace%files(1) = 2
        return
    end if

    ! Set cycle state point properties.
    recomp_cycle%temp = temp
    recomp_cycle%pres = pres
    recomp_cycle%enth = enth
    recomp_cycle%entr = entr
    recomp_cycle%dens = dens

    ! Calculate performance metrics for low-temperature recuperator.
    recomp_cycle%LT%C_dot_hot = m_dot_t * (enth(8) - enth(9)) / (temp(8) - temp(9))    ! LT recuperator hot stream capacitance rate
    recomp_cycle%LT%C_dot_cold = m_dot_mc * (enth(3) - enth(2)) / (temp(3) - temp(2))  ! LT recuperator cold stream capacitance rate
    C_dot_min = min(recomp_cycle%LT%C_dot_hot, recomp_cycle%LT%C_dot_cold)
    Q_dot_max = C_dot_min * (temp(8) - temp(2))
    recomp_cycle%LT%eff = Q_dot_LT / Q_dot_max  ! definition of effectiveness
    recomp_cycle%LT%UA_design = UA_LT_calc
    recomp_cycle%LT%UA = UA_LT_calc
    recomp_cycle%LT%DP_design = [pres(2) - pres(3), pres(8) - pres(9)]
    recomp_cycle%LT%m_dot_design = [m_dot_mc, m_dot_t]
    recomp_cycle%LT%Q_dot = Q_dot_LT
    recomp_cycle%LT%min_DT = min_DT_LT
    recomp_cycle%LT%N_sub = N_sub_hxrs

    ! Calculate performance metrics for high-temperature recuperator.
    recomp_cycle%HT%C_dot_hot = m_dot_t * (enth(7) - enth(8)) / (temp(7) - temp(8))   ! HT recuperator hot stream capacitance rate
    recomp_cycle%HT%C_dot_cold = m_dot_t * (enth(5) - enth(4)) / (temp(5) - temp(4))  ! HT recuperator cold stream capacitance rate
    C_dot_min = min(recomp_cycle%HT%C_dot_hot, recomp_cycle%HT%C_dot_cold)
    Q_dot_max = C_dot_min * (temp(7) - temp(4))
    recomp_cycle%HT%eff = Q_dot_HT / Q_dot_max  ! definition of effectiveness
    recomp_cycle%HT%UA_design = UA_HT_calc
    recomp_cycle%HT%UA = UA_HT_calc
    recomp_cycle%HT%DP_design = [pres(4) - pres(5), pres(7) - pres(8)]
    recomp_cycle%HT%m_dot_design = [m_dot_t, m_dot_t]
    recomp_cycle%HT%Q_dot = Q_dot_HT
    recomp_cycle%HT%min_DT = min_DT_HT
    recomp_cycle%HT%N_sub = N_sub_hxrs

    ! Set relevant values for other heat exchangers.
    recomp_cycle%PHX%Q_dot = m_dot_t * (enth(6) - enth(5))
    recomp_cycle%PHX%DP_design = [pres(5) - pres(6), 0.0_dp]
    recomp_cycle%PHX%m_dot_design = [m_dot_t, 0.0_dp]
    recomp_cycle%PC%Q_dot = m_dot_mc * (enth(9) - enth(1))
    recomp_cycle%PC%DP_design = [0.0_dp, pres(9) - pres(1)]
    recomp_cycle%PC%m_dot_design = [0.0_dp, m_dot_mc]

    ! Calculate cycle performance metrics.
    recomp_cycle%recomp_frac = recomp_frac
    recomp_cycle%W_dot_net = w_mc * m_dot_mc + w_rc * m_dot_rc + w_t * m_dot_t
    recomp_cycle%eta_thermal = recomp_cycle%W_dot_net / recomp_cycle%PHX%Q_dot
    recomp_cycle%m_dot_turbine = m_dot_t
    recomp_cycle%conv_tol = tol

end subroutine design
```

```
subroutine optimal_design( &
    W_dot_net,             &  ! [input] target net cycle power (kW)
    T_mc_in,               &  ! [input] compressor inlet temperature (K)
    T_t_in,                &  ! [input] turbine inlet temperature (K)
    DP_LT,                 &  ! [input] pressure drops in low-temperature recuperator (kPa if positive values)
    DP_HT,                 &  ! [input] pressure drops in high-temperature recuperator (kPa if positive values)
    DP_PC,                 &  ! [input] pressure drops in precooler (kPa if positive values)
    DP_PHX,                &  ! [input] pressure drops in primary heat exchanger (kPa if positive values)
    UA_rec_total,          &  ! [input] total design-point recuperator UA value (kW/K)
    eta_mc,                &  ! [input] design-point efficiency of the main compressor; isentropic if positive, polytropic if negative
    eta_rc,                &  ! [input] design-point efficiency of the recompressor; isentropic if positive, polytropic if negative
    eta_t,                 &  ! [input] design-point efficiency of the turbine; isentropic if positive, polytropic if negative
    N_sub_hxrs,            &  ! [input] number of sub-heat exchangers to use when calculating UA value for a heat exchanger
    P_high_limit,          &  ! [input] maximum allowable pressure in cycle (kPa)
    P_mc_out_guess,        &  ! [input] initial guess for main compressor outlet pressure (kPa)
    fixed_P_mc_out,        &  ! [input] if .true., P_mc_out is fixed at P_mc_out_guess
    PR_mc_guess,           &  ! [input] initial guess for ratio of P_mc_out to P_mc_in (-)
    fixed_PR_mc,           &  ! [input] if .true., ratio of P_mc_out to P_mc_in is fixed at PR_mc_guess
    recomp_frac_guess,     &  ! [input] initial guess for design-point recompression fraction
    fixed_recomp_frac,     &  ! [input] if .true., recomp_frac is fixed at recomp_frac_guess
    LT_frac_guess,         &  ! [input] initial guess for fraction of UA_rec_total that is in the low-temperature recuperator
    fixed_LT_frac,         &  ! [input] if .true., LT_frac is fixed at LT_frac_guess
    tol,                   &  ! [input] cycle convergence tolerance
    opt_tol,               &  ! [input] optimization convergence tolerance
    error_trace,           &  ! [output] an ErrorTrace object
    recomp_cycle           &  ! [output] a RecompCycle object
    )

    ! Arguments
    real(dp), intent(in) :: W_dot_net, T_mc_in, T_t_in, UA_rec_total, eta_mc, eta_rc, eta_t
    real(dp), intent(in) :: P_high_limit, P_mc_out_guess, PR_mc_guess, recomp_frac_guess, LT_frac_guess, tol, opt_tol
    logical, intent(in)  :: fixed_P_mc_out, fixed_PR_mc, fixed_recomp_frac, fixed_LT_frac
    integer, intent(in)  :: N_sub_hxrs
    real(dp), dimension(2), intent(in) :: DP_LT, DP_HT, DP_PC, DP_PHX
    type(ErrorTrace), intent(out)  :: error_trace
    type(RecompCycle), intent(out) :: recomp_cycle

    ! Subplex Parameters and Variables
    integer, parameter :: maxf = 200
    integer, parameter :: max_free_vars = 4
    integer, parameter :: mode = 0
    integer  :: iflag, iwork(50), nfe
    real(dp) :: subplex_fmin, scale(max_free_vars), work(50), x(max_free_vars)

    ! Local Variables
    type(RecompCycle) :: opt_recomp_cycle
    integer  :: n, index
    logical  :: solution_found

    ! Initialize guess array.
    x = 0.0_dp
    index = 1
    if (.not. fixed_P_mc_out) then
        x(index) = P_mc_out_guess
        scale(index) = 500.0_dp  ! pressure scale
        index = index + 1
    end if
    if (.not. fixed_PR_mc) then
        x(index) = PR_mc_guess
        scale(index) = 0.2_dp  ! pressure ratio scale
        index = index + 1
    end if
    if (.not. fixed_recomp_frac) then
        x(index) = recomp_frac_guess
        scale(index) = 0.05_dp  ! recompression fraction scale
        index = index + 1
    end if
    if (.not. fixed_LT_frac) then
        x(index) = LT_frac_guess
        scale(index) = 0.05_dp  ! recuperator split scale
        index = index + 1
    end if
    n = index - 1

    ! Call subplex if any inputs can vary, or just call the design subroutine
    if (n > 0) then  ! call subplex
        solution_found = .false.
        opt_recomp_cycle%eta_thermal = 0.0_dp  ! ensure thermal efficiency is initialized to 0 (should be, but just to be sure)
        call subplx(design_point_eta, n, opt_tol, maxf, mode, scale, x, subplex_fmin, nfe, work, iwork, iflag)
        if (solution_found) then
            recomp_cycle = opt_recomp_cycle
        else
            error_trace%code = 1
            error_trace%lines(1) = 711
            error_trace%files(1) = 2
        end if
    else  ! no inputs vary; just call design subroutine
        call design(                                    &
            W_dot_net = W_dot_net,                      &
            T_mc_in = T_mc_in,                          &
            T_t_in = T_t_in,                            &
```

```
        P_mc_in = P_mc_out_guess / PR_mc_guess,          &
        P_mc_out = P_mc_out_guess,                       &
        DP_LT = DP_LT,                                   &
        DP_HT = DP_HT,                                   &
        DP_PC = DP_PC,                                   &
        DP_PHX = DP_PHX,                                 &
        UA_LT = UA_rec_total * LT_frac_guess,            &
        UA_HT = UA_rec_total * (1.0_dp - LT_frac_guess), &
        recomp_frac = recomp_frac_guess,                 &
        eta_mc = eta_mc,                                 &
        eta_rc = eta_rc,                                 &
        eta_t = eta_t,                                   &
        N_sub_hxrs = N_sub_hxrs,                         &
        tol = tol,                                       &
        error_trace = error_trace,                       &
        recomp_cycle = recomp_cycle                      &
        )
    if (error_trace%code /= 0) then
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 720
        error_trace%files(index) = 2
    end if
end if
recomp_cycle%high_pressure_limit = P_high_limit  ! store high pressure limit

contains

    real(dp) function design_point_eta(n, x)
        ! Call the design subroutine with inputs contained in the x array.  Other required inputs are
        ! passed transparently because of the scope.
        integer, intent(in)  :: n     ! number of inputs that are varied during optimization
        real(dp), intent(in) :: x(n)  ! inputs with order: P_mc_out, PR_mc, recomp_frac, LT_frac (some can be missing)
        real(dp) :: P_mc_in_local, P_mc_out_local, PR_mc_local, recomp_frac_local, LT_frac_local

        ! Extract input variables from x.
        index = 1
        if (.not. fixed_P_mc_out) then
            P_mc_out_local = x(index)
            index = index + 1
        else
            P_mc_out_local = P_mc_out_guess
        end if
        if (.not. fixed_PR_mc) then
            PR_mc_local = x(index)
            index = index + 1
        else
            PR_mc_local = PR_mc_guess
        end if
        P_mc_in_local = P_mc_out_local / PR_mc_local
        if (.not. fixed_recomp_frac) then
            recomp_frac_local = x(index)
            index = index + 1
        else
            recomp_frac_local = recomp_frac_guess
        end if
        if (.not. fixed_LT_frac) then
            LT_frac_local = x(index)
            index = index + 1
        else
            LT_frac_local = LT_frac_guess
        end if

        ! Check inputs.
        if (recomp_frac_local < 0.0_dp) then
            design_point_eta = 0.0_dp
            return
        end if
        if (LT_frac_local < 0.0_dp .or. LT_frac_local > 1.0_dp) then
            design_point_eta = 0.0_dp
            return
        end if
        if (P_mc_out_local > P_high_limit) then
            design_point_eta = 0.0_dp
            return
        end if
        if (P_mc_in_local >= P_mc_out_local) then
            design_point_eta = 0.0_dp
            return
        end if
        if (P_mc_in_local < 100.0_dp) then  ! low-pressure limit
            design_point_eta = 0.0_dp
            return
        end if
        if (PR_mc_local > 50.0_dp) then  ! pressure ratio limit
            design_point_eta = 0.0_dp
            return
        end if

        ! Call design subroutine.
        call design(                                     &
            W_dot_net = W_dot_net,                       &
            T_mc_in = T_mc_in,                           &
```

```
                T_t_in = T_t_in,                                &
                P_mc_in = P_mc_in_local,                        &
                P_mc_out = P_mc_out_local,                      &
                DP_LT = DP_LT,                                  &
                DP_HT = DP_HT,                                  &
                DP_PC = DP_PC,                                  &
                DP_PHX = DP_PHX,                                &
                UA_LT = UA_rec_total * LT_frac_local,           &
                UA_HT = UA_rec_total * (1.0_dp - LT_frac_local), &
                recomp_frac = recomp_frac_local,                &
                eta_mc = eta_mc,                                &
                eta_rc = eta_rc,                                &
                eta_t = eta_t,                                  &
                N_sub_hxrs = N_sub_hxrs,                        &
                tol = tol,                                      &
                error_trace = error_trace,                      &
                recomp_cycle = recomp_cycle                     &
                )
            if (error_trace%code == 0) then
                design_point_eta = -recomp_cycle%eta_thermal  ! subplex is a minimizer, so return negative efficiency
                if (recomp_cycle%eta_thermal > opt_recomp_cycle%eta_thermal) then
                    solution_found = .true.
                    opt_recomp_cycle = recomp_cycle
                end if
            else
                design_point_eta = 0.0_dp
            end if

        end function design_point_eta

end subroutine optimal_design


subroutine auto_optimal_design( &
    W_dot_net,                  &  ! [input] target net cycle power (kW)
    T_mc_in,                    &  ! [input] compressor inlet temperature (K)
    T_t_in,                     &  ! [input] turbine inlet temperature (K)
    DP_LT,                      &  ! [input] pressure drops in low-temperature recuperator (kPa if positive values)
    DP_HT,                      &  ! [input] pressure drops in high-temperature recuperator (kPa if positive values)
    DP_PC,                      &  ! [input] pressure drops in precooler (kPa if positive values)
    DP_PHX,                     &  ! [input] pressure drops in primary heat exchanger (kPa if positive values)
    UA_rec_total,               &  ! [input] total design-point recuperator UA value (kW/K)
    eta_mc,                     &  ! [input] design-point efficiency of the main compressor; isentropic if positive, polytropic if negative
    eta_rc,                     &  ! [input] design-point efficiency of the recompressor; isentropic if positive, polytropic if negative
    eta_t,                      &  ! [input] design-point efficiency of the turbine; isentropic if positive, polytropic if negative
    N_sub_hxrs,                 &  ! [input] number of sub-heat exchangers to use when calculating UA value for a heat exchanger
    P_high_limit,               &  ! [input] maximum allowable pressure in cycle (kPa)
    tol,                        &  ! [input] cycle convergence tolerance
    opt_tol,                    &  ! [input] optimization convergence tolerance
    error_trace,                &  ! [output] an ErrorTrace object
    recomp_cycle                &  ! [output] a RecompCycle object
    )

    ! Arguments
    real(dp), intent(in) :: W_dot_net, T_mc_in, T_t_in, UA_rec_total, eta_mc, eta_rc, eta_t, P_high_limit, tol, opt_tol
    integer, intent(in)  :: N_sub_hxrs
    real(dp), dimension(2), intent(in) :: DP_LT, DP_HT, DP_PC, DP_PHX
    type(ErrorTrace), intent(out)  :: error_trace
    type(RecompCycle), intent(out) :: recomp_cycle

    ! Local Variables
    real(dp) :: best_P_high, PR_mc_guess
    type(RecompCycle) :: test_cycle

    ! External Functions
    real(dp), external :: fmin

    ! Outer optimization loop (best results are stored in recomp_cycle).
    recomp_cycle%eta_thermal = 0.0_dp  ! initialize for optimization loop
    best_P_high = fmin(P_high_limit*0.2_dp, P_high_limit, opt_eta, 1.0_dp)

    ! Check model with P_mc_out set at P_high_limit for a recompression and simple cycle and use the better configuration.
    PR_mc_guess = recomp_cycle%pres(2) / recomp_cycle%pres(1)  ! optimal pressure ratio from outer optimization loop
    call optimal_design(                &  ! recompression cycle
        W_dot_net = W_dot_net,          &
        T_mc_in = T_mc_in,              &
        T_t_in = T_t_in,                &
        DP_LT = DP_LT,                  &
        DP_HT = DP_HT,                  &
        DP_PC = DP_PC,                  &
        DP_PHX = DP_PHX,                &
        UA_rec_total = UA_rec_total,    &
        eta_mc = eta_mc,                &
        eta_rc = eta_rc,                &
        eta_t = eta_t,                  &
        N_sub_hxrs = N_sub_hxrs,        &
        P_high_limit = P_high_limit,    &
        P_mc_out_guess = P_high_limit,  &
        fixed_P_mc_out = .true.,        &
        PR_mc_guess = PR_mc_guess,      &
        fixed_PR_mc = .false.,          &
        recomp_frac_guess = 0.3_dp,     &
```

```
    fixed_recomp_frac = .false.,     &
    LT_frac_guess = 0.5_dp,          &
    fixed_LT_frac = .false.,         &
    tol = tol,                       &
    opt_tol = opt_tol,               &
    error_trace = error_trace,       &
    recomp_cycle = test_cycle        &
    )
if (error_trace%code == 0 .and. test_cycle%eta_thermal >= recomp_cycle%eta_thermal) recomp_cycle = test_cycle
call optimal_design(                 &  ! simple cycle
    W_dot_net = W_dot_net,           &
    T_mc_in = T_mc_in,               &
    T_t_in = T_t_in,                 &
    DP_LT = DP_LT,                   &
    DP_HT = DP_HT,                   &
    DP_PC = DP_PC,                   &
    DP_PHX = DP_PHX,                 &
    UA_rec_total = UA_rec_total,     &
    eta_mc = eta_mc,                 &
    eta_rc = eta_rc,                 &
    eta_t = eta_t,                   &
    N_sub_hxrs = N_sub_hxrs,         &
    P_high_limit = P_high_limit,     &
    P_mc_out_guess = P_high_limit,   &
    fixed_P_mc_out = .true.,         &
    PR_mc_guess = PR_mc_guess,       &
    fixed_PR_mc = .false.,           &
    recomp_frac_guess = 0.0_dp,      &
    fixed_recomp_frac = .true.,      &
    LT_frac_guess = 0.5_dp,          &
    fixed_LT_frac = .true.,          &
    tol = tol,                       &
    opt_tol = opt_tol,               &
    error_trace = error_trace,       &
    recomp_cycle = test_cycle        &
    )
if (error_trace%code == 0 .and. test_cycle%eta_thermal >= recomp_cycle%eta_thermal) recomp_cycle = test_cycle

contains

    real(dp) function opt_eta(P_high)
        ! Call the optimal_design subroutine with fixed P_high.  Other required inputs are
        ! passed transparently because of the scope.
        real(dp), intent(in) :: P_high
        type(RecompCycle) :: local_simple_cycle, local_recomp_cycle
        if (P_high > P_pseudocritical(T_mc_in)) then  ! start with P_mc_in at pseudocritical pressure
            PR_mc_guess = P_high / P_pseudocritical(T_mc_in)
        else
            PR_mc_guess = 1.1_dp
        end if

        call optimal_design(             &  ! recompression cycle
            W_dot_net = W_dot_net,       &
            T_mc_in = T_mc_in,           &
            T_t_in = T_t_in,             &
            DP_LT = DP_LT,               &
            DP_HT = DP_HT,               &
            DP_PC = DP_PC,               &
            DP_PHX = DP_PHX,             &
            UA_rec_total = UA_rec_total, &
            eta_mc = eta_mc,             &
            eta_rc = eta_rc,             &
            eta_t = eta_t,               &
            N_sub_hxrs = N_sub_hxrs,     &
            P_high_limit = P_high_limit, &
            P_mc_out_guess = P_high,     &
            fixed_P_mc_out = .true.,     &
            PR_mc_guess = PR_mc_guess,   &
            fixed_PR_mc = .false.,       &
            recomp_frac_guess = 0.3_dp,  &
            fixed_recomp_frac = .false., &
            LT_frac_guess = 0.5_dp,      &
            fixed_LT_frac = .false.,     &
            tol = tol,                   &
            opt_tol = opt_tol,           &
            error_trace = error_trace,   &
            recomp_cycle = local_recomp_cycle &
            )
        if (error_trace%code == 0 .and. local_recomp_cycle%eta_thermal >= recomp_cycle%eta_thermal) then
            recomp_cycle = local_recomp_cycle
        end if

        call optimal_design(             &  ! simple cycle
            W_dot_net = W_dot_net,       &
            T_mc_in = T_mc_in,           &
            T_t_in = T_t_in,             &
            DP_LT = DP_LT,               &
            DP_HT = DP_HT,               &
            DP_PC = DP_PC,               &
            DP_PHX = DP_PHX,             &
            UA_rec_total = UA_rec_total, &
            eta_mc = eta_mc,             &
```

```
                eta_rc = eta_rc,                &
                eta_t = eta_t,                  &
                N_sub_hxrs = N_sub_hxrs,        &
                P_high_limit = P_high_limit,    &
                P_mc_out_guess = P_high,        &
                fixed_P_mc_out = .true.,        &
                PR_mc_guess = PR_mc_guess,      &
                fixed_PR_mc = .false.,          &
                recomp_frac_guess = 0.0_dp,     &
                fixed_recomp_frac = .true.,     &
                LT_frac_guess = 0.5_dp,         &
                fixed_LT_frac = .true.,         &
                tol = tol,                      &
                opt_tol = opt_tol,              &
                error_trace = error_trace,      &
                recomp_cycle = local_simple_cycle &
                )
            if (error_trace%code == 0 .and. local_simple_cycle%eta_thermal >= recomp_cycle%eta_thermal) then
                recomp_cycle = local_simple_cycle
            end if

            opt_eta = -max(local_recomp_cycle%eta_thermal, local_simple_cycle%eta_thermal)  ! fmin is a minimizer, so return a negative value

        end function opt_eta

end subroutine auto_optimal_design


real(dp) function P_pseudocritical(T)
    ! Return the approximate pseudocritical pressure (kPa) as a function of
    ! temperature (K) for carbon dioxide using a curve fit.
    real(dp), intent(in) :: T
    P_pseudocritical = (0.191448_dp * T + 45.6661_dp) * T - 24213.3_dp
end function P_pseudocritical


end module design_point
```

Appendix III: off_design_point Module

off_design_point.f90

```
!-------------------------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-------------------------------------------------------------------------------------------------------------------------------------------
!
! This file contains the the module 'off_design_point', which defines five system-level subroutines:
!   off_design -- the main off-design cycle model
!   target_off_design -- given a power or heat addition target, iterates on P1 to match it (returns an error if target not possible)
!   target_off_design_alt -- given a power or heat addition target, iterates on P1 to match it or return the nearest possible
!   optimal_off_design -- iterates on inputs to off_design in order to maximize power output or thermal efficiency
!   optimal_target_off_design -- determines the maximum efficiency of the cycle for a give target
!
! Notes:
!   1) The optimization routines often need a bit of tweaking (initial guess values, variable scales, etc.) and improvements
!      could be made w.r.t. handling invalid inputs.  There are likely better techniques that could be developed in
!      order to provide fast and stable optimization (especially in regards to the optimal_target_off_design subroutine).
!   2) Allowing surge and supersonic tip speeds makes convergence easier and the optimal results typically end of being valid, so
!      the values for the parameters 'surge_allowed' and 'supersonic_tip_speed_allowed' are .true. by default.
!   3) The optimal_target_off_design subroutine uses max W_dot_net as proxy for max_Q_dot if the target is not possible.
!   4) The target_off_design and optimal_target_off_design subroutines allow high-side pressures slightly above the high-pressure
!      limit to help the optimization routines; calls to these subroutines should be checked for satisfactory pressures.
!
! Cycle State Points:
!   1)  mc in           / PC out
!   2)  LT in (cold)    / mc out
!   3)  mixing valve in / LT out (cold)
!   4)  HT in (cold)    / mixing valve out
!   5)  PHX in          / HT out (cold)
!   6)  turbine in      / PHX out
!   7)  HT in (hot)     / turbine out
!   8)  LT in (hot)     / HT out (hot)
!   9)  PC and rc in    / LT out (hot)
!   10) mixing valve in / recomp out
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: August 20, 2014
!
!-------------------------------------------------------------------------------------------------------------------------------------------

module off_design_point

use core
use compressors
use turbines
use heat_exchangers
implicit none
private
public :: off_design, target_off_design, optimal_off_design, optimal_target_off_design

logical, parameter :: surge_allowed = .true.
logical, parameter :: supersonic_tip_speed_allowed = .true.


contains


subroutine off_design( &
    recomp_cycle,      &  ! [input/output] a RecompCycle object with design-point variables set
    T_mc_in,           &  ! [input] compressor inlet temperature (K)
    T_t_in,            &  ! [input] turbine inlet temperature (K)
    P_mc_in,           &  ! [input] compressor inlet pressure (kPa)
    recomp_frac,       &  ! [input] fraction of flow that bypasses the precooler and main compressor
    N_mc,              &  ! [input] main compressor shaft speed (rpm)
    N_t,               &  ! [input] turbine shaft speed (rpm)
    N_sub_hxrs,        &  ! [input] number of sub-heat exchangers to use when calculating UA value for a heat exchanger
    tol,               &  ! [input] convergence tolerance
    error_trace        &  ! [output] an ErrorTrace object
    )

    use CO2_Properties, only: CO2_TP, CO2_PH
```

```
! Arguments
type(RecompCycle), intent(inout) :: recomp_cycle
real(dp), intent(in) :: T_mc_in, T_t_in, P_mc_in, recomp_frac, N_mc, N_t, tol
integer, intent(in)  :: N_sub_hxrs
type(ErrorTrace), intent(out) :: error_trace

! Parameters
integer, parameter :: max_iter = 100
real(dp), parameter :: temperature_tolerance = 1.0e-6_dp  ! temperature differences below this are considered zero

! Local Variables
integer  :: m_dot_iter, T9_iter, T8_iter, error_code, index
real(dp) :: rho_in, C_dot_min, Q_dot_max, m_dot_t_allowed, m_dot_residual, partial_phi, tip_speed
real(dp) :: m_dot_lower_bound, m_dot_upper_bound, m_dot_mc_guess, m_dot_mc_max, last_m_dot_guess, last_m_dot_residual
real(dp) :: T9_lower_bound, T9_upper_bound, T8_lower_bound, T8_upper_bound, last_LT_residual, last_T9_guess
real(dp) :: last_HT_residual, last_T8_guess, secant_guess
real(dp) :: m_dot_t, m_dot_mc, m_dot_rc, UA_LT, UA_HT, w_mc, w_rc, w_t
real(dp) :: min_DT_LT, min_DT_HT, UA_LT_calc, UA_HT_calc, Q_dot_LT, Q_dot_HT, UA_HT_residual, UA_LT_residual
real(dp), dimension(10) :: temp, pres, enth, entr, dens
real(dp), dimension(2) :: DP_LT, DP_HT, DP_PC, DP_PHX
logical :: first_pass

! Initialize a few variables.
temp(1) = T_mc_in
pres(1) = P_mc_in
temp(6) = T_t_in
recomp_cycle%mc%N = N_mc
recomp_cycle%t%N = N_t
recomp_cycle%conv_tol = tol

! Prepare the mass flow rate iteration loop.
call CO2_TP(T=temp(1), P=pres(1), error_code=error_code, dens=rho_in)
if (error_code /= 0) then
    error_trace%code = error_code
    error_trace%lines(1) = 113
    error_trace%files(1) = 3
    return
end if
tip_speed = recomp_cycle%mc%D_rotor * 0.5_dp * N_mc * 0.10471975512_dp  ! main compressor tip speed in m/s
partial_phi = rho_in * recomp_cycle%mc%D_rotor**2 * tip_speed           ! reduces computation on next two lines
m_dot_mc_guess = recomp_cycle%mc%phi_design * partial_phi               ! mass flow rate corresponding to design-point phi in ...
m_dot_mc_max = recomp_cycle%mc%phi_max * partial_phi * 1.2_dp           ! largest possible mass flow rate in main compressor (with ...
m_dot_t = m_dot_mc_guess / (1.0_dp - recomp_frac)                       ! first guess for mass flow rate through turbine
m_dot_upper_bound = m_dot_mc_max / (1.0_dp - recomp_frac)               ! largest possible mass flow rate through turbine
m_dot_lower_bound = 0.0_dp                                              ! this lower bound allows for surge (checked after iteration)
first_pass = .true.

! Enter the mass flow rate iteration loop.
m_dot_loop: do m_dot_iter = 1, max_iter
    m_dot_rc = m_dot_t * recomp_frac  ! mass flow rate through recompressing compressor
    m_dot_mc = m_dot_t - m_dot_rc     ! mass flow rate through compressor

    ! Calculate the pressure rise through the main compressor.
    call off_design_compressor(    &
        comp = recomp_cycle%mc,    &
        T_in = temp(1),            &
        P_in = pres(1),            &
        m_dot = m_dot_mc,          &
        N = N_mc,                  &
        error_trace = error_trace, &
        T_out = temp(2),           &
        P_out = pres(2)            &
        )
    if (error_trace%code == 1) then  ! m_dot is too high because the given shaft speed is not possible
        m_dot_upper_bound = m_dot_t
        m_dot_t = (m_dot_lower_bound + m_dot_upper_bound) * 0.5_dp  ! use bisection for new mass flow rate guess
        cycle
    else if (error_trace%code == 2) then  ! m_dot is too low because P_out is (likely) above properties limits
        m_dot_lower_bound = m_dot_t
        m_dot_t = (m_dot_lower_bound + m_dot_upper_bound) * 0.5_dp  ! use bisection for new mass flow rate guess
        cycle
    else if (error_trace%code /= 0) then  ! unexpected error
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 135
        error_trace%files(index) = 3
        return
    end if

    ! Calculate scaled pressure drops through heat exchangers.
    DP_LT  = hxr_pressure_drops(hxr=recomp_cycle%LT, m_dots=[m_dot_mc, m_dot_t])
    DP_HT  = hxr_pressure_drops(hxr=recomp_cycle%HT, m_dots=[m_dot_t, m_dot_t])
    DP_PHX = hxr_pressure_drops(hxr=recomp_cycle%PHX, m_dots=[m_dot_t, 0.0_dp])  ! not concerned with hot stream of PHX
    DP_PC  = hxr_pressure_drops(hxr=recomp_cycle%PC, m_dots=[0.0_dp, m_dot_mc])  ! not concerned with cold stream of precooler

    ! Apply pressure drops to heat exchangers, fully defining the pressures at all states.
    pres(3)  = pres(2) - DP_LT(1)   ! LT recuperator (cold stream)
    pres(4)  = pres(3)              ! assume no pressure drop in mixing valve
    pres(10) = pres(3)              ! assume no pressure drop in mixing valve
    pres(5)  = pres(4) - DP_HT(1)   ! HT recuperator (cold stream)
    pres(6)  = pres(5) - DP_PHX(1)  ! PHX
    pres(9)  = pres(1) + DP_PC(2)   ! precooler
    pres(8)  = pres(9) + DP_LT(2)   ! LT recuperator (hot stream)
```

```
    pres(7)  = pres(8) + DP_HT(2)   ! HT recuperator (hot stream)

    ! Calculate the mass flow rate through the turbine.
    call off_design_turbine(       &
        turb = recomp_cycle%t,     &
        T_in = temp(6),            &
        P_in = pres(6),            &
        P_out = pres(7),           &
        N = N_t,                   &
        error_trace = error_trace, &
        m_dot = m_dot_t_allowed,   &
        T_out = temp(7)            &
        )
    if (error_trace%code /= 0) then  ! unexpected error
        index = next_trace_index(error_trace)
        error_trace%lines(index) = 177
        error_trace%files(index) = 3
        return
    end if

    ! Determine the mass flow rate residual and prepare the next iteration.
    m_dot_residual = m_dot_t - m_dot_t_allowed
    secant_guess = m_dot_t - m_dot_residual * (last_m_dot_guess - m_dot_t) / (last_m_dot_residual - m_dot_residual)  ! next guess ...
    if (m_dot_residual > 0.0_dp) then  ! pressure rise is too small, so m_dot_t is too big
        if (m_dot_residual / m_dot_t < tol) exit m_dot_loop  ! residual is positive; check for convergence
        m_dot_upper_bound = m_dot_t   ! reset upper bound
    else  ! pressure rise is too high, so m_dot_t is too small
        if (-m_dot_residual / m_dot_t < tol) exit m_dot_loop ! residual is negative; check for convergence
        m_dot_lower_bound = m_dot_t   ! reset lower bound
    end if
    last_m_dot_residual = m_dot_residual                              ! reset last stored residual value
    last_m_dot_guess = m_dot_t                                  ! reset last stored guess value

    ! Check if the secant method overshoots and fall back to bisection if it does.
    if (first_pass) then
        m_dot_t = (m_dot_upper_bound + m_dot_lower_bound) * 0.5_dp
        first_pass = .false.
    else if (secant_guess < m_dot_lower_bound .or. secant_guess > m_dot_upper_bound) then  ! secant method overshot, use bisection
        m_dot_t = (m_dot_upper_bound + m_dot_lower_bound) * 0.5_dp
    else
        m_dot_t = secant_guess
    end if

end do m_dot_loop

! Check for convergence.
if (m_dot_iter >= max_iter) then
    error_trace%code = 42
    error_trace%lines(1) = 220
    error_trace%files(1) = 3
    return
end if

! Fully define known states.
call CO2_TP(T=temp(1), P=pres(1), error_code=error_code, enth=enth(1), entr=entr(1), dens=dens(1))
if (error_code /= 0) then
    error_trace%code = error_code
    error_trace%lines(1) = 228
    error_trace%files(1) = 3
    return
end if
call CO2_TP(T=temp(2), P=pres(2), error_code=error_code, enth=enth(2), entr=entr(2), dens=dens(2))
if (error_code /= 0) then
    error_trace%code = error_code
    error_trace%lines(1) = 235
    error_trace%files(1) = 3
    return
end if
call CO2_TP(T=temp(6), P=pres(6), error_code=error_code, enth=enth(6), entr=entr(6), dens=dens(6))
if (error_code /= 0) then
    error_trace%code = error_code
    error_trace%lines(1) = 242
    error_trace%files(1) = 3
    return
end if
call CO2_TP(T=temp(7), P=pres(7), error_code=error_code, enth=enth(7), entr=entr(7), dens=dens(7))
if (error_code /= 0) then
    error_trace%code = error_code
    error_trace%lines(1) = 249
    error_trace%files(1) = 3
    return
end if

! Get the recuperator conductances corresponding to the converged mass flow rates.
UA_LT = hxr_conductance(hxr=recomp_cycle%LT, m_dots=[m_dot_mc, m_dot_t])
UA_HT = hxr_conductance(hxr=recomp_cycle%HT, m_dots=[m_dot_t, m_dot_t])

! Outer iteration loop: temp(8), checking against UA_HT.
if (UA_HT < 1.0e-12_dp) then  ! no high-temperature recuperator
    T8_lower_bound = temp(7)  ! no iteration necessary
    T8_upper_bound = temp(7)  ! no iteration necessary
    temp(8) = temp(7)
```

```
    UA_HT_calc = 0.0_dp
    last_HT_residual = 0.0_dp
    last_T8_guess = temp(7)
else
    T8_lower_bound = temp(2)    ! the absolute lowest temp(8) could be
    T8_upper_bound = temp(7)    ! the absolutely highest temp(8) could be
    temp(8) = (T8_lower_bound + T8_upper_bound) * 0.5_dp  ! bisect bounds for first guess
    UA_HT_calc = -1.0_dp
    last_HT_residual = UA_HT    ! know a priori that with T8 = T7, UA_calc = 0 therefore residual is UA_HT - 0.0
    last_T8_guess = temp(7)
end if
T8_loop: do T8_iter = 1,max_iter

    ! Fully define state 8.
    call CO2_TP(T=temp(8), P=pres(8), error_code=error_code, enth=enth(8), entr=entr(8), dens=dens(8))
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 280
        error_trace%files(1) = 3
        return
    end if

    ! Inner iteration loop: temp(9), checking against UA_LT.
    if (UA_LT < 1.0e-12_dp) then  ! no low-temperature recuperator
        T9_lower_bound = temp(8)  ! no iteration necessary
        T9_upper_bound = temp(8)  ! no iteration necessary
        temp(9) = temp(8)
        UA_LT_calc = 0.0_dp
        last_LT_residual = 0.0_dp
        last_T9_guess = temp(8)
    else
        T9_lower_bound = temp(2)    ! the absolute lowest temp(9) could be
        T9_upper_bound = temp(8)    ! the absolutely highest temp(9) could be
        temp(9) = (T9_lower_bound + T9_upper_bound) * 0.5_dp  ! bisect bounds for first guess
        UA_LT_calc = -1.0_dp
        last_LT_residual = UA_LT    ! know a priori that with T9 = T8, UA_calc = 0 therefore residual is UA_LT - 0
        last_T9_guess = temp(8)
    end if
    T9_loop: do T9_iter = 1,max_iter

            call CO2_TP(T=temp(9), P=pres(9), error_code=error_code, enth=enth(9), entr=entr(9), dens=dens(9))  ! fully define state 9
            if (error_code /= 0) then
                error_trace%code = error_code
                error_trace%lines(1) = 306
                error_trace%files(1) = 3
                return
            end if

            if (recomp_frac >= 1.0e-12_dp) then  ! determine the required shaft speed for the recompressing compressor
                call off_design_recompressor(  &
                    comp = recomp_cycle%rc,    &
                    T_in = temp(9),            &
                    P_in = pres(9),            &
                    m_dot = m_dot_rc,          &
                    P_out = pres(10),          &
                    error_trace = error_trace, &
                    T_out = temp(10)           &
                    )
                if (error_trace%code /= 0) then
                    index = next_trace_index(error_trace)
                    error_trace%lines(index) = 315
                    error_trace%files(index) = 3
                    return
                end if
                call CO2_TP(T=temp(10), P=pres(10), error_code=error_code, enth=enth(10), entr=entr(10), dens=dens(10))  ! fully ...
                if (error_code /= 0) then
                    error_trace%code = error_code
                    error_trace%lines(1) = 330
                    error_trace%files(1) = 3
                    return
                end if
            else
                temp(10) = temp(9)  ! assume state 10 is the same as state 9
                enth(10) = enth(9)
                entr(10) = entr(9)
                dens(10) = dens(9)
            end if

            ! Calculate the UA value of the low-temperature recuperator.
            if (UA_LT < 1.0e-12_dp) then  ! no low-temp recuperator (this check is necessary to prevent pressure drops with UA=0 ...
                Q_dot_LT = 0.0_dp
            else
                Q_dot_LT = m_dot_t * (enth(8) - enth(9))
            end if
            call calculate_hxr_UA(         &
                N_sub_hxrs = N_sub_hxrs,   &
                Q_dot = Q_dot_LT,          &
                m_dot_c = m_dot_mc,        &
                m_dot_h = m_dot_t,         &
                T_c_in = temp(2),          &
                T_h_in = temp(8),          &
                P_c_in = pres(2),          &
```

```
            P_c_out = pres(3),         &
            P_h_in = pres(8),          &
            P_h_out = pres(9),         &
            error_trace = error_trace, &
            UA = UA_LT_calc,           &
            min_DT = min_DT_LT         &
            )
        if (error_trace%code > 0) then
            if (error_trace%code == 11) then  ! second-law violation in hxr, therefore temp(9) is too low
                T9_lower_bound = temp(9)
                temp(9) = (T9_lower_bound + T9_upper_bound) * 0.5_dp  ! bisect bounds for next guess
                error_trace%code = 0  ! reset error trace
                error_trace%lines = 0
                error_trace%files = 0
                cycle T9_loop
            else
                index = next_trace_index(error_trace)
                error_trace%lines(index) = 350
                error_trace%files(index) = 3
                return
            end if
        end if

        ! Check for convergence and adjust T9 appropriately.
        UA_LT_residual = UA_LT - UA_LT_calc
        if (abs(UA_LT_residual) < 1.0e-12_dp) exit T9_loop  ! catches no LT case
        secant_guess = temp(9) - UA_LT_residual * (last_T9_guess - temp(9)) / (last_LT_residual - UA_LT_residual)  ! next guess ...
        if (UA_LT_residual < 0.0_dp) then  ! UA_LT_calc is too big, temp(9) needs to be higher
            if (abs(UA_LT_residual)/UA_LT < tol) exit T9_loop  ! UA_LT converged (residual is negative)
            T9_lower_bound = temp(9)
        else  ! UA_LT_calc is too small, temp(9) needs to be lower
            if (UA_LT_residual/UA_LT < tol) exit T9_loop  ! UA_LT converged
            if (min_DT_LT < temperature_tolerance) exit T9_loop  ! UA_calc is still too low but there isn't anywhere to go so ...
            T9_upper_bound = temp(9)
        end if
        last_LT_residual = UA_LT_residual  ! reset last stored residual value
        last_T9_guess = temp(9)  ! reset last stored guess value

        ! Check if the secant method overshoots and fall back to bisection if it does.
        if (secant_guess <= T9_lower_bound .or. secant_guess >= T9_upper_bound .or. secant_guess /= secant_guess) then  ! secant ...
            temp(9) = (T9_lower_bound + T9_upper_bound) * 0.5_dp
        else
            temp(9) = secant_guess
        end if

end do T9_loop

! Check that T9_loop converged.
if (T9_iter >= max_iter) then
    error_trace%code = 31
    error_trace%lines(1) = 406
    error_trace%files(1) = 3
    return
end if

! State 3 can now be fully defined.
enth(3) = enth(2) + Q_dot_LT / m_dot_mc  ! energy balance on cold stream of low-temp recuperator
call CO2_PH(P=pres(3), H=enth(3), error_code=error_code, temp=temp(3), entr=entr(3), dens=dens(3))
if (error_code /= 0) then
    error_trace%code = error_code
    error_trace%lines(1) = 415
    error_trace%files(1) = 3
    return
end if

! Go through mixing valve.
if (recomp_frac >= 1.0e-12_dp) then
    enth(4) = (1.0_dp - recomp_frac) * enth(3) + recomp_frac * enth(10)  ! conservation of energy (both sides divided by m_dot_t)
    call CO2_PH(P=pres(4), H=enth(4), error_code=error_code, temp=temp(4), entr=entr(4), dens=dens(4))
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 426
        error_trace%files(1) = 3
        return
    end if
else  ! no mixing valve, therefore state 4 is equal to state 3
    temp(4) = temp(3)
    enth(4) = enth(3)
    entr(4) = entr(3)
    dens(4) = dens(3)
end if

! Check for a second law violation at the outlet of the high-temp recuperator.
if (temp(4) >= temp(8)) then  ! temp(8) is not valid and it must be increased
    T8_lower_bound = temp(8)
    temp(8) = (T8_lower_bound + T8_upper_bound) * 0.5_dp
    cycle T8_loop
end if

! Calculate the UA value of the high-temperature recuperator.
if (UA_HT < 1.0e-12_dp) then  ! no high-temp recuperator (this check is necessary to prevent pressure drops with UA=0 from ...
    Q_dot_HT = 0.0_dp
```

```
    else
        Q_dot_HT = m_dot_t * (enth(7) - enth(8))
    end if
    call calculate_hxr_UA(         &
        N_sub_hxrs = N_sub_hxrs,   &
        Q_dot = Q_dot_HT,          &
        m_dot_c = m_dot_t,         &
        m_dot_h = m_dot_t,         &
        T_c_in = temp(4),          &
        T_h_in = temp(7),          &
        P_c_in = pres(4),          &
        P_c_out = pres(5),         &
        P_h_in = pres(7),          &
        P_h_out = pres(8),         &
        error_trace = error_trace, &
        UA = UA_HT_calc,           &
        min_DT = min_DT_HT         &
        )
    if (error_trace%code > 0) then
        if (error_trace%code == 11) then  ! second-law violation in hxr, therefore temp(8) is too low
            T8_lower_bound = temp(8)
            temp(8) = (T8_lower_bound + T8_upper_bound) * 0.5_dp  ! bisect bounds for next guess
            error_trace%code = 0  ! reset error trace
            error_trace%lines = 0
            error_trace%files = 0
            cycle T8_loop
        else
            index = next_trace_index(error_trace)
            error_trace%lines(index) = 453
            error_trace%files(index) = 3
            return
        end if
    end if

    ! Check for convergence and adjust T8 appropriately.
    UA_HT_residual = UA_HT - UA_HT_calc
    if (abs(UA_HT_residual) < 1.0e-12_dp) exit T8_loop  ! catches no HT case
    secant_guess = temp(8) - UA_HT_residual * (last_T8_guess - temp(8)) / (last_HT_residual - UA_HT_residual)  ! next guess predicted ...
    if (UA_HT_residual < 0.0_dp) then  ! UA_HT_calc is too big, temp(8) needs to be higher
        if (abs(UA_HT_residual)/UA_HT < tol) exit T8_loop  ! UA_HT converged (residual is negative)
        T8_lower_bound = temp(8)
    else  ! UA_HT_calc is too small, temp(8) needs to be lower
        if (UA_HT_residual/UA_HT < tol) exit T8_loop  ! UA_HT converged
        if (min_DT_HT < temperature_tolerance) exit T8_loop  ! UA_calc is still too low but there isn't anywhere to go so it's ok ...
        T8_upper_bound = temp(8)
    end if
    last_HT_residual = UA_HT_residual  ! reset last stored residual value
    last_T8_guess = temp(8)  ! reset last stored guess value

    ! Check if the secant method overshoots and fall back to bisection if it does.
    if (secant_guess <= T8_lower_bound .or. secant_guess >= T8_upper_bound) then  ! secant method overshot, use bisection
        temp(8) = (T8_lower_bound + T8_upper_bound) * 0.5_dp
    else
        temp(8) = secant_guess
    end if

end do T8_loop

! Check that T8_loop converged.
if (T8_iter >= max_iter) then
    error_trace%code = 35
    error_trace%lines(1) = 509
    error_trace%files(1) = 3
    return
end if

! State 5 can now be fully defined.
enth(5) = enth(4) + Q_dot_HT / m_dot_t  ! energy balance on cold stream of high-temp recuperator
call CO2_PH(P=pres(5), H=enth(5), error_code=error_code, temp=temp(5), entr=entr(5), dens=dens(5))
if (error_code /= 0) then
    error_trace%code = error_code
    error_trace%lines(1) = 518
    error_trace%files(1) = 3
    return
end if

! Set cycle state point properties.
recomp_cycle%temp = temp
recomp_cycle%pres = pres
recomp_cycle%enth = enth
recomp_cycle%entr = entr
recomp_cycle%dens = dens

! Calculate performance metrics for low-temperature recuperator.
recomp_cycle%LT%C_dot_hot = m_dot_t * (enth(8) - enth(9)) / (temp(8) - temp(9))    ! LT recuperator hot stream capacitance rate
recomp_cycle%LT%C_dot_cold = m_dot_mc * (enth(3) - enth(2)) / (temp(3) - temp(2))  ! LT recuperator cold stream capacitance rate
C_dot_min = min(recomp_cycle%LT%C_dot_hot, recomp_cycle%LT%C_dot_cold)
Q_dot_max = C_dot_min * (temp(8) - temp(2))
recomp_cycle%LT%eff = Q_dot_LT / Q_dot_max  ! definition of effectiveness
recomp_cycle%LT%Q_dot = Q_dot_LT
recomp_cycle%LT%min_DT = min_DT_LT
recomp_cycle%LT%N_sub = N_sub_hxrs
```

```
    ! Calculate performance metrics for high-temperature recuperator.
    recomp_cycle%HT%C_dot_hot = m_dot_t * (enth(7) - enth(8)) / (temp(7) - temp(8))   ! HT recuperator hot stream capacitance rate
    recomp_cycle%HT%C_dot_cold = m_dot_t * (enth(5) - enth(4)) / (temp(5) - temp(4))  ! HT recuperator cold stream capacitance rate
    C_dot_min = min(recomp_cycle%HT%C_dot_hot, recomp_cycle%HT%C_dot_cold)
    Q_dot_max = C_dot_min * (temp(7) - temp(4))
    recomp_cycle%HT%eff = Q_dot_HT / Q_dot_max  ! definition of effectiveness
    recomp_cycle%HT%UA_design = UA_HT_calc
    recomp_cycle%HT%DP_design = [pres(4) - pres(5), pres(7) - pres(8)]
    recomp_cycle%HT%m_dot_design = [m_dot_t, m_dot_t]
    recomp_cycle%HT%Q_dot = Q_dot_HT
    recomp_cycle%HT%min_DT = min_DT_HT
    recomp_cycle%HT%N_sub = N_sub_hxrs

    ! Set relevant values for other heat exchangers.
    recomp_cycle%PHX%Q_dot = m_dot_t * (enth(6) - enth(5))
    recomp_cycle%PC%Q_dot = m_dot_mc * (enth(9) - enth(1))

    ! Calculate cycle performance metrics.
    w_mc = enth(1) - enth(2)  ! specific work of compressor (kJ/kg) [negative]
    w_t = enth(6) - enth(7)   ! specific work of turbine (kJ/kg) [positive]
    if (recomp_frac > 0.0_dp) then
        w_rc = enth(9) - enth(10)  ! specific work of recompressor (kJ/kg) [negative]
    else
        w_rc = 0.0_dp
    end if
    recomp_cycle%W_dot_net = w_mc * m_dot_mc + w_rc * m_dot_rc + w_t * m_dot_t
    recomp_cycle%eta_thermal = recomp_cycle%W_dot_net / recomp_cycle%PHX%Q_dot
    recomp_cycle%recomp_frac = recomp_frac
    recomp_cycle%m_dot_turbine = m_dot_t

end subroutine off_design


subroutine target_off_design(  &
    recomp_cycle,              &  ! [input/output] a RecompCycle object with design-point variables set
    T_mc_in,                   &  ! [input] compressor inlet temperature (K)
    T_t_in,                    &  ! [input] turbine inlet temperature (K)
    recomp_frac,               &  ! [input] recompression fraction
    N_mc,                      &  ! [input] main compressor shaft speed
    N_t,                       &  ! [input] turbine shaft speed
    target,                    &  ! [input] what value to aim for
    target_code,               &  ! [input] type of target: 1) W_dot 2) Q_dot_PHX
    lowest_pressure,           &  ! [input] lowest pressure to check
    highest_pressure,          &  ! [input] highest pressure to check
    N_sub_hxrs,                &  ! [input] number of sub-heat exchangers to use when calculating UA value for a heat exchanger
    tol,                       &  ! [input] convergence tolerance
    error_trace                &  ! [output] an ErrorTrace object
    )

    ! Given a target and a target_code, iterate on pressure to match the target.  This subroutine returns an error if the
    ! target is not possible given the other inputs.  Alternatively, 'target_off_design_alt' subroutine can be used to match the
    ! target or get as close as possible to the value. (e.g., if the target is 10 MW but the maximum power output of the cycle is
    ! 8.5 MW, the alternative subroutine will return 8.5 MW, while this subroutine will return an error.)

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    real(dp), intent(in) :: T_mc_in, T_t_in, recomp_frac, N_mc, N_t, target, lowest_pressure, highest_pressure, tol
    integer, intent(in)  :: target_code, N_sub_hxrs
    type(ErrorTrace), intent(out) :: error_trace

    ! Parameters
    integer, parameter :: max_iter = 100
    integer, parameter :: search_intervals = 20  ! number of intervals to check for valid bounds before starting secant loop

    ! Local Variables
    type(RecompCycle) :: biggest_cycle
    real(dp) :: P_low, P_high, P_guess, left_residual, right_residual, residual, last_P_guess, last_residual, P_secant
    real(dp) :: target_value, biggest_value
    real(dp), dimension(0:search_intervals) :: P_guesses
    integer :: i, iter
    logical :: lower_bound_found, upper_bound_found

    ! Determine the interval containing the solution.
    lower_bound_found = .false.
    upper_bound_found = .false.
    left_residual = -1.0e12_dp  ! initialized to large negative value
    right_residual = 1.0e12_dp  ! initialized to large positive value
    P_low = lowest_pressure
    P_high = highest_pressure
    P_guesses = [ ( P_low + i * (P_high - P_low) / real(search_intervals,dp) , i = 0, search_intervals ) ]  ! create linear vector of guesses
    biggest_value = 0.0_dp
    biggest_cycle = recomp_cycle
    do i = 0, search_intervals
        P_guess = P_guesses(i)
        call off_design(               &
            recomp_cycle = recomp_cycle, &
            T_mc_in = T_mc_in,         &
            T_t_in = T_t_in,           &
            P_mc_in = P_guess,         &
            recomp_frac = recomp_frac,   &
            N_mc = N_mc,               &
```

```
        N_t = N_t,                  &
        N_sub_hxrs = N_sub_hxrs,    &
        tol = tol,                  &
        error_trace = error_trace   &
        )
    if (error_trace%code == 0) then
        if (recomp_cycle%pres(2) > recomp_cycle%high_pressure_limit * 1.2_dp) exit  ! compressor inlet pressure is getting too big
        select case (target_code)
            case (1); target_value = recomp_cycle%W_dot_net
            case (2); target_value = recomp_cycle%PHX%Q_dot
        end select
        residual = target_value - target
        if (target_value > biggest_value) then  ! keep track of the largest value seen
            biggest_cycle = recomp_cycle
            biggest_value = target_value
        end if
        if (residual >= 0.0_dp) then  ! value is above target
            if (residual < right_residual) then  ! first right bound or a better bound; use it
                P_high = P_guess
                right_residual = residual
                upper_bound_found = .true.
            end if
        else  ! value is below target
            if (residual > left_residual) then  ! note: residual and left_residual are negative
                P_low = P_guess
                left_residual = residual
                lower_bound_found = .true.
            end if
        end if
    end if
    if (lower_bound_found .and. upper_bound_found) exit
end do

if (.not. (lower_bound_found .and. upper_bound_found)) then  ! solution not found in interval; return cycle with largest target
    error_trace%code = 26  ! this is a specific code that is used by optimal_target_off_design
    error_trace%lines(1) = 667
    error_trace%files(1) = 3
    recomp_cycle = biggest_cycle
    return
end if

! Enter secant / bisection loop.
P_guess = (P_low + P_high) * 0.5_dp  ! start with bisection (note: could use left and right bounds and residuals to get a better...
do iter = 1, max_iter

    call off_design(                &
        recomp_cycle = recomp_cycle, &
        T_mc_in = T_mc_in,          &
        T_t_in = T_t_in,            &
        P_mc_in = P_guess,          &
        recomp_frac = recomp_frac,  &
        N_mc = N_mc,                &
        N_t = N_t,                  &
        N_sub_hxrs = N_sub_hxrs,    &
        tol = tol,                  &
        error_trace = error_trace   &
        )
    if (error_trace%code /= 0) then  ! results not valid; choose a random value between P_low and P_high for next guess
        call random_number(P_guess)  ! 0 <= P_guess < 1
        P_guess = P_low + (P_high - P_low) * P_guess
        cycle
    end if

    ! Check residual
    select case (target_code)
        case (1); residual = recomp_cycle%W_dot_net - target  ! W_dot
        case (2); residual = recomp_cycle%PHX%Q_dot - target  ! Q_dot_PHX
    end select
    if (residual >= 0.0_dp) then  ! value is above target
        if (residual / target <= tol) exit  ! converged
        P_high = P_guess
    else  ! value is below target
        if (-residual / target <= tol) exit  ! converged (residual is negative)
        P_low = P_guess
    end if
    if (abs(P_high-P_low) < 0.1_dp) exit  ! interval is tiny; consider it converged

    ! Determine next guess.
    P_secant = P_guess - residual * (last_P_guess - P_guess) / (last_residual - residual)  ! next guess predicted using secant method
    last_P_guess = P_guess
    last_residual = residual
    P_guess = P_secant
    if (P_guess <= P_low .or. P_guess >= P_high) P_guess = (P_low + P_high) * 0.5_dp  ! secant overshot, use bisection

end do

! Check for convergence.
if (iter >= max_iter) then
    error_trace%code = 82
    error_trace%lines(1) = 721
    error_trace%files(1) = 3
    return
```

```
    end if

end subroutine target_off_design


subroutine target_off_design_alt( &
    recomp_cycle,              &  ! [input/output] a RecompCycle object with design-point variables set
    T_mc_in,                   &  ! [input] compressor inlet temperature (K)
    T_t_in,                    &  ! [input] turbine inlet temperature (K)
    recomp_frac,               &  ! [input] recompression fraction
    N_mc,                      &  ! [input] main compressor shaft speed
    N_t,                       &  ! [input] turbine shaft speed
    target,                    &  ! [input] what value to aim for
    target_code,               &  ! [input] type of target: 1) W_dot 2) Q_dot_PHX
    lowest_pressure,           &  ! [input] lowest pressure to check
    highest_pressure,          &  ! [input] highest pressure to check
    N_sub_hxrs,                &  ! [input] number of sub-heat exchangers to use when calculating UA value for a heat exchanger
    tol,                       &  ! [input] convergence tolerance
    error_trace                &  ! [output] an ErrorTrace object
    )

    ! Given a target and a target_code, iterate on pressure to find the minimum residual between the actual and calculated.

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    real(dp), intent(in) :: T_mc_in, T_t_in, recomp_frac, N_mc, N_t, target, lowest_pressure, highest_pressure, tol
    integer, intent(in)  :: target_code, N_sub_hxrs
    type(ErrorTrace), intent(out) :: error_trace

    ! Parameters
    real(dp), parameter :: fmin_tol = 0.01_dp  ! absolute pressure tolerance

    ! External Functions
    real(dp), external :: fmin

    ! Local Variables
    type(RecompCycle) :: best_recomp_cycle
    real(dp) :: best_residual
    logical  :: solution_found

    solution_found = .false.
    best_residual = 1.0e12_dp
    best_residual = fmin(lowest_pressure, highest_pressure, target_residual, fmin_tol)
    if (solution_found) then
        recomp_cycle = best_recomp_cycle
    else
        error_trace%code = 999
        error_trace%lines(1) = 768
        error_trace%files(1) = 3
    end if

    contains

        real(dp) function target_residual(P_mc_in)
            ! Return the absolute value of the residual between the target and
            ! its calculated value.  No validity checking is performed.
            real(dp), intent(in) :: P_mc_in

            call off_design(                &
                recomp_cycle = recomp_cycle, &
                T_mc_in = T_mc_in,           &
                T_t_in = T_t_in,             &
                P_mc_in = P_mc_in,           &
                recomp_frac = recomp_frac,   &
                N_mc = N_mc,                 &
                N_t = N_t,                   &
                N_sub_hxrs = N_sub_hxrs,     &
                tol = tol,                   &
                error_trace = error_trace    &
                )
            if (error_trace%code /= 0) then
                target_residual = 1.0e15_dp
                return
            end if

            select case (target_code)
                case (1); target_residual = recomp_cycle%W_dot_net - target  ! W_dot
                case (2); target_residual = recomp_cycle%PHX%Q_dot - target  ! Q_dot_PHX
            end select

            if (abs(target_residual) < abs(best_residual)) then
                solution_found = .true.
                best_residual = target_residual
                best_recomp_cycle = recomp_cycle
            end if

            target_residual = abs(target_residual)

        end function target_residual

end subroutine target_off_design_alt
```

```
subroutine optimal_off_design( &
    recomp_cycle,              &  ! [input/output] a RecompCycle object with design-point variables set
    T_mc_in,                   &  ! [input] compressor inlet temperature (K)
    T_t_in,                    &  ! [input] turbine inlet temperature (K)
    value_code,                &  ! [input] value to maximize: 1) eta, 2) W_dot
    N_sub_hxrs,                &  ! [input] number of sub-heat exchangers to use when calculating UA value for a heat exchanger
    P_mc_in_guess,             &  ! [input] initial guess for P_mc_in when iterating to hit target, or set P_mc_in if value_code is 0
    fixed_P_mc_in,             &  ! [input] if .true., P_mc_in is fixed at P_mc_in_guess
    recomp_frac_guess,         &  ! [input] initial guess for recompression fraction
    fixed_recomp_frac,         &  ! [input] if .true., recomp_frac is fixed at recomp_frac_guess
    N_mc_guess,                &  ! [input] initial guess for main compressor shaft speed
    fixed_N_mc,                &  ! [input] if .true., N_mc is fixed at N_mc_guess
    N_t_guess,                 &  ! [input] initial guess for turbine shaft speed (negative value links it to N_mc)
    fixed_N_t,                 &  ! [input] if .true., N_t is fixed at N_t_guess
    tol,                       &  ! [input] convergence tolerance
    opt_tol,                   &  ! [input] optimization convergence tolerance
    error_trace                &  ! [output] an ErrorTrace object
    )

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    real(dp), intent(in) :: T_mc_in, T_t_in, P_mc_in_guess, recomp_frac_guess, N_mc_guess, N_t_guess, tol, opt_tol
    logical, intent(in)  :: fixed_P_mc_in, fixed_recomp_frac, fixed_N_mc, fixed_N_t
    integer, intent(in)  :: value_code, N_sub_hxrs
    type(ErrorTrace), intent(out)  :: error_trace

    ! Subplex Parameters and Variables
    integer, parameter :: maxf = 200
    integer, parameter :: max_free_vars = 4
    integer  :: iflag, iwork(50), mode, nfe
    real(dp) :: fmin, scale(max_free_vars), work(50), x(max_free_vars)

    ! Local Variables
    real(dp) :: largest_value, N_t_local
    integer  :: n, index
    logical  :: solution_found
    type(RecompCycle) :: optimal_cycle

    ! Initialize guess array.
    x = 0.0_dp
    index = 1
    if (.not. fixed_P_mc_in) then
        x(index) = P_mc_in_guess
        scale(index) = 50.0_dp  ! P_mc_in scale
        index = index + 1
    end if
    if (.not. fixed_recomp_frac) then
        x(index) = recomp_frac_guess
        scale(index) = 0.01_dp   ! recomp scale
        index = index + 1
    end if
    if (.not. fixed_N_mc) then
        x(index) = N_mc_guess
        scale(index) = 100.0_dp   ! N_mc_scale
        index = index + 1
    end if
    if (.not. fixed_N_t) then
        x(index) = N_t_guess
        scale(index) = 100.0_dp   ! N_t_scale
        index = index + 1
    end if
    n = index - 1

    if (n > 0) then  ! need to call subplex
        solution_found = .false.
        largest_value = 0.0_dp
        mode = 0
        call subplx(off_design_point_value, n, opt_tol, maxf, mode, scale, x, fmin, nfe, work, iwork, iflag)
        if (solution_found) then
            recomp_cycle = optimal_cycle
            error_trace%code = 0
            error_trace%lines = 0
            error_trace%files = 0
        else
            error_trace%code = 111
            error_trace%lines(1) = 886
            error_trace%files(1) = 3
            return
        end if
    else  ! just call off_design subroutine (with fixed inputs)
        if (N_t_guess <= 0.0_dp) then
            N_t_local = N_mc_guess  ! link turbine and main compressor shafts
        else
            N_t_local = N_t_guess
        end if
        call off_design(                    &
            recomp_cycle = recomp_cycle,    &
            T_mc_in = T_mc_in,              &
            T_t_in = T_t_in,                &
            P_mc_in = P_mc_in_guess,        &
            recomp_frac = recomp_frac_guess, &
```

```
        N_mc = N_mc_guess,               &
        N_t = N_t_local,                 &
        N_sub_hxrs = N_sub_hxrs,         &
        tol = tol,                       &
        error_trace = error_trace        &
        )
    if (error_trace%code == 0) then  ! check validity of results
        solution_found = recomp_cycle%pres(2) <= recomp_cycle%high_pressure_limit  ! high-pressure limit
        if (.not. surge_allowed) then
            if (recomp_cycle%mc%surge) solution_found = .false.
            if (recomp_cycle%recomp_frac > 0.0_dp .and. recomp_cycle%rc%surge) solution_found = .false.
        end if
        if (.not. supersonic_tip_speed_allowed) then
            if (recomp_cycle%mc%w_tip_ratio > 1.0_dp) solution_found = .false.
            if (recomp_cycle%recomp_frac > 0.0_dp .and. recomp_cycle%rc%w_tip_ratio > 1.0_dp) solution_found = .false.
            if (recomp_cycle%t%w_tip_ratio > 1.0_dp) solution_found = .false.
        end if
        if (.not. solution_found) then
            error_trace%code = 112
            error_trace%lines(1) = 904
            error_trace%files(1) = 3
        end if
    end if
end if

contains

    real(dp) function off_design_point_value(n, x)
        ! Call the off_design subroutine with inputs contained in the x array.
        ! Returns the power output or thermal efficiency, depending on the value code.
        integer, intent(in)  :: n     ! number of inputs that are varied during optimization
        real(dp), intent(in) :: x(n)  ! inputs with order: recomp_frac, N_mc, N_t (some can be missing)
        real(dp) :: P_mc_in_local, recomp_frac_local, N_mc_local, N_t_local

        ! Extract input variables from x.
        index = 1
        if (.not. fixed_P_mc_in) then
            P_mc_in_local = x(index)
            index = index + 1
        else
            P_mc_in_local = P_mc_in_guess
        end if
        if (.not. fixed_recomp_frac) then
            recomp_frac_local = x(index)
            index = index + 1
        else
            recomp_frac_local = recomp_frac_guess
        end if
        if (.not. fixed_N_mc) then
            N_mc_local = x(index)
            index = index + 1
        else
            N_mc_local = N_mc_guess
        end if
        if (.not. fixed_N_t) then
            N_t_local = x(index)
            index = index + 1
        else
            N_t_local = N_t_guess
        end if
        if (N_t_local <= 0.0_dp) N_t_local = N_mc_local  ! link turbine and main compressor shafts

        ! Check inputs.
        if (recomp_frac_local < 0.0_dp) then
            off_design_point_value = 0.0_dp
            return
        end if

        ! Call off_design subroutine.
        call off_design(                    &
            recomp_cycle = recomp_cycle,    &
            T_mc_in = T_mc_in,              &
            T_t_in = T_t_in,                &
            P_mc_in = P_mc_in_local,        &
            recomp_frac = recomp_frac_local, &
            N_mc = N_mc_local,              &
            N_t = N_t_local,                &
            N_sub_hxrs = N_sub_hxrs,        &
            tol = tol,                      &
            error_trace = error_trace       &
            )
        if (error_trace%code /= 0) then
            off_design_point_value = 0.0_dp
            return
        end if
        select case (value_code)
            case (1); off_design_point_value = -recomp_cycle%eta_thermal
            case (2); off_design_point_value = -recomp_cycle%W_dot_net
        end select

        ! Check validity.
        if (recomp_cycle%pres(2) > recomp_cycle%high_pressure_limit) then  ! above high-pressure limit; provide optimizer with ...
```

```
                off_design_point_value = off_design_point_value / (10_dp + recomp_cycle%pres(2) - recomp_cycle%high_pressure_limit)
            end if
            if (.not. surge_allowed) then
                if (recomp_cycle%mc%surge) off_design_point_value = 0.0_dp
                if (recomp_cycle%recomp_frac > 0.0_dp .and. recomp_cycle%rc%surge) off_design_point_value = 0.0_dp
            end if
            if (.not. supersonic_tip_speed_allowed) then
                if (recomp_cycle%mc%w_tip_ratio > 1.0_dp) off_design_point_value = 0.0_dp
                if (recomp_cycle%recomp_frac > 0.0_dp .and. recomp_cycle%rc%w_tip_ratio > 1.0_dp) off_design_point_value = 0.0_dp
                if (recomp_cycle%t%w_tip_ratio > 1.0_dp) off_design_point_value = 0.0_dp
            end if

            ! Check if this is the optimal cycle.
            if (abs(off_design_point_value) > largest_value) then
                solution_found = .true.
                optimal_cycle = recomp_cycle
                largest_value = abs(off_design_point_value)
            end if

        end function off_design_point_value

end subroutine optimal_off_design


subroutine optimal_target_off_design( &
    recomp_cycle,                     &  ! [input/output] a RecompCycle object with design-point variables set
    T_mc_in,                          &  ! [input] compressor inlet temperature (K)
    T_t_in,                           &  ! [input] turbine inlet temperature (K)
    target,                           &  ! [input] target value for W_dot_net or Q_dot_PHX (kW)
    target_code,                      &  ! [input] type of optimization: 1) target W_dot (max eta), 2) target Q_dot_PHX (max eta)
    N_sub_hxrs,                       &  ! [input] number of sub-heat exchangers to use when calculating UA value for a hxr
    lowest_pressure,                  &  ! [input] the lowest pressure to check
    highest_pressure,                 &  ! [input] the highest pressure to check
    recomp_frac_guess,                &  ! [input] initial guess for recompression fraction
    fixed_recomp_frac,                &  ! [input] if .true., recomp_frac is fixed at recomp_frac_guess
    N_mc_guess,                       &  ! [input] initial guess for main compressor shaft speed
    fixed_N_mc,                       &  ! [input] if .true., N_mc is fixed at N_mc_guess
    N_t_guess,                        &  ! [input] initial guess for turbine shaft speed (negative value links it to N_mc)
    fixed_N_t,                        &  ! [input] if .true., N_t is fixed at N_t_guess
    tol,                              &  ! [input] convergence tolerance
    opt_tol,                          &  ! [input] optimization convergence tolerance
    error_trace                       &  ! [output] an ErrorTrace object
    )

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    real(dp), intent(in) :: T_mc_in, T_t_in, target, lowest_pressure, highest_pressure, recomp_frac_guess, N_mc_guess, N_t_guess
    real(dp), intent(in) :: tol, opt_tol
    logical, intent(in)  :: fixed_recomp_frac, fixed_N_mc, fixed_N_t
    integer, intent(in)  :: target_code, N_sub_hxrs
    type(ErrorTrace), intent(out)  :: error_trace

    ! Subplex Parameters and Variables
    integer, parameter :: maxf = 200
    integer, parameter :: max_free_vars = 3
    integer  :: iflag, iwork(50), mode, nfe
    real(dp) :: subplex_fmin, scale(max_free_vars), work(50), x(max_free_vars)

    ! Local Variables
    type(RecompCycle) :: best_recomp_cycle
    real(dp) :: best_eta, biggest_target, P_low, unused_var
    integer  :: index, n
    logical  :: solution_found, point_found

    ! Determine the largest possible power output of the cycle.
    point_found = .false.
    P_low = lowest_pressure
    do
        call optimal_off_design(                  &
            recomp_cycle = recomp_cycle,          &
            T_mc_in = T_mc_in,                    &
            T_t_in = T_t_in,                      &
            value_code = 2,                       &  ! max W_dot
            N_sub_hxrs = N_sub_hxrs,              &
            P_mc_in_guess = P_low,                &
            fixed_P_mc_in = .false.,              &
            recomp_frac_guess = recomp_frac_guess, &
            fixed_recomp_frac = fixed_recomp_frac, &
            N_mc_guess = N_mc_guess,              &
            fixed_N_mc = fixed_N_mc,              &
            N_t_guess = N_t_guess,                &
            fixed_N_t = fixed_N_t,                &
            tol = tol,                            &
            opt_tol = opt_tol,                    &
            error_trace = error_trace             &
            )
        if (error_trace%code == 0) then
            if (point_found) exit  ! exit only after testing two starting points (prevents optimization near-misses)
            point_found = .true.
        end if
        P_low = P_low + 500.0_dp
        if (P_low > highest_pressure) exit
```

```
end do

if (.not. point_found) then  ! this is an unexpected error
    error_trace%code = 99
    error_trace%lines(1) = 1096
    error_trace%files(1) = 3
    return
end if

select case (target_code)
    case (1); biggest_target = recomp_cycle%W_dot_net
    case (2); biggest_target = recomp_cycle%PHX%Q_dot
end select

! If the target is not possible, return the cycle with the largest (based on power output).
if (biggest_target <= target) then
    error_trace%code = 0  ! reset error code
    error_trace%lines = 0
    error_trace%files = 0
    return
end if

! Initialize guess array.
x = 0.0_dp
index = 1
if (.not. fixed_recomp_frac) then
    x(index) = recomp_frac_guess
    scale(index) = 0.01_dp   ! recomp scale
    index = index + 1
end if
if (.not. fixed_N_mc) then
    x(index) = N_mc_guess
    scale(index) = 100.0_dp   ! N_mc_scale
    index = index + 1
end if
if (.not. fixed_N_t) then
    x(index) = N_t_guess
    scale(index) = 100.0_dp   ! N_t_scale
    index = index + 1
end if
n = index - 1
solution_found = .false.
best_eta = 0.0_dp
if (n > 0) then  ! call subplex
    mode = 0
    call subplx(eta_at_target, n, opt_tol, maxf, mode, scale, x, subplex_fmin, nfe, work, iwork, iflag)
else
    unused_var = eta_at_target(n, x)  ! necessary to get recomp_cycle at target (ignores x array) [warning: somewhat untested]
end if
if (.not. solution_found) then
    error_trace%code = 98
    error_trace%lines(1) = 1143
    error_trace%files(1) = 3
    return
end if

return

contains

    real(dp) function eta_at_target(n, x)
        ! Call the target_off_design subroutine with inputs contained in the x array.
        ! Returns (negative) thermal efficiency.
        integer, intent(in)  :: n     ! number of inputs that are varied during optimization
        real(dp), intent(in) :: x(n)  ! inputs with order: recomp_frac, N_mc, N_t (some can be missing)
        real(dp) :: recomp_frac_local, N_mc_local, N_t_local

        ! Extract input variables from x.
        index = 1
        if (.not. fixed_recomp_frac) then
            recomp_frac_local = x(index)
            index = index + 1
        else
            recomp_frac_local = recomp_frac_guess
        end if
        if (.not. fixed_N_mc) then
            N_mc_local = x(index)
            index = index + 1
        else
            N_mc_local = N_mc_guess
        end if
        if (.not. fixed_N_t) then
            N_t_local = x(index)
            index = index + 1
        else
            N_t_local = N_t_guess
        end if
        if (N_t_local <= 0.0_dp) N_t_local = N_mc_local  ! link turbine and main compressor shafts if necessary

        ! Check inputs.
        if (recomp_frac_local < 0.0_dp) then
            eta_at_target = 0.0_dp
```

```
                return
            end if

            ! Call target_off_design subroutine.
            call target_off_design(                  &
                recomp_cycle = recomp_cycle,         &
                T_mc_in = T_mc_in,                   &
                T_t_in = T_t_in,                     &
                recomp_frac = recomp_frac_local,     &
                N_mc = N_mc_local,                   &
                N_t = N_t_local,                     &
                target = target,                     &
                target_code = target_code,           &
                lowest_pressure = lowest_pressure,   &
                highest_pressure = highest_pressure, &
                N_sub_hxrs = N_sub_hxrs,             &
                tol = tol,                           &
                error_trace = error_trace            &
                )
            if (error_trace%code == 26) then  ! could not hit target
                eta_at_target = 1.0_dp / (100.0_dp + abs(recomp_cycle%W_dot_net))  ! provides a directional hint to optimizer
                return
            else if (error_trace%code /= 0) then  ! uncaught error
                eta_at_target = 0.0_dp
                return
            else
                eta_at_target = recomp_cycle%eta_thermal
            end if

            ! Check validity.
            if (recomp_cycle%pres(2) > recomp_cycle%high_pressure_limit) then
                eta_at_target = eta_at_target / (10.0_dp + recomp_cycle%pres(2) - recomp_cycle%high_pressure_limit)  ! provides a ...
            end if
            if (.not. surge_allowed) then
                if (recomp_cycle%mc%surge) eta_at_target = 0.0_dp
                if (recomp_cycle%recomp_frac > 0.0_dp .and. recomp_cycle%rc%surge) eta_at_target = 0.0_dp
            end if
            if (.not. supersonic_tip_speed_allowed) then
                if (recomp_cycle%mc%w_tip_ratio > 1.0_dp) eta_at_target = 0.0_dp
                if (recomp_cycle%recomp_frac > 0.0_dp .and. recomp_cycle%rc%w_tip_ratio > 1.0_dp) eta_at_target = 0.0_dp
                if (recomp_cycle%t%w_tip_ratio > 1.0_dp) eta_at_target = 0.0_dp
            end if

            ! Check if this is the best solution.
            if (eta_at_target > best_eta) then
                best_eta = eta_at_target
                best_recomp_cycle = recomp_cycle
                solution_found = .true.
            end if

            eta_at_target = -eta_at_target  ! subplex is minimizer

        end function eta_at_target

end subroutine optimal_target_off_design


end module off_design_point
```

Appendix IV: compressors Module

snl_compressor.f90

```
!-------------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-------------------------------------------------------------------------------------------------------------------------------
!
! This file contains the the module 'compressors', which defines a number of subroutines based on the radial compressor being
! studied at Sandia National Laboratory.
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: July 12, 2014
!
!-------------------------------------------------------------------------------------------------------------------------------

module compressors

use core
implicit none
private
public :: compressor_sizing, recompressor_sizing, off_design_compressor, off_design_recompressor

real(dp), parameter :: snl_phi_design = 0.02971_dp  ! design-point flow coefficient for Sandia compressor (corresponds to max eta)
real(dp), parameter :: snl_phi_min = 0.02_dp        ! approximate surge limit for SNL compressor
real(dp), parameter :: snl_phi_max = 0.05_dp        ! approximate x-intercept for SNL compressor


contains


subroutine compressor_sizing(recomp_cycle, error_trace)
    ! Determine the compressor rotor diameter and design-point shaft speed
    ! and store values in recomp_cycle%mc.
    !
    ! Arguments:
    !   recomp_cycle -- a RecompCycle object that defines the simple/recompression cycle at the design point
    !   error_trace -- an ErrorTrace object

    use CO2_properties, only: CO2_TD, CO2_PS

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    type(ErrorTrace), intent(out) :: error_trace

    ! Local Variables
    integer  :: error_code
    real(dp) :: D_in, h_in, s_in, T_out, P_out, h_out, D_out, ssnd_out, h_s_out, psi_design, m_dot, w_i, U_tip, N_rad_s

    ! Create references to cycle state properties for clarity.
    D_in = recomp_cycle%dens(1)
    h_in = recomp_cycle%enth(1)
    s_in = recomp_cycle%entr(1)
    T_out = recomp_cycle%temp(2)
    P_out = recomp_cycle%pres(2)
    h_out = recomp_cycle%enth(2)
    D_out = recomp_cycle%dens(2)
    call CO2_TD(T=T_out, D=D_out, error_code=error_code, ssnd=ssnd_out)  ! speed of sound at outlet
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 65
        error_trace%files(1) = 4
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet specific enthalpy after isentropic compression
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 72
        error_trace%files(1) = 4
        return
    end if

    ! Calculate psi at the design-point phi using Horner's method
    psi_design = ((((-498626.0_dp * snl_phi_design) + 53224.0_dp) * snl_phi_design - 2505.0_dp) * snl_phi_design + 54.6_dp) &
                 * snl_phi_design + 0.04049_dp  ! from dimensionless modified head curve (at design-point, psi and modified psi are equal)
```

```
    ! Determine required size and speed of compressor.
    m_dot = recomp_cycle%m_dot_turbine * (1.0_dp - recomp_cycle%recomp_frac)  ! mass flow rate through compressor (kg/s)
    w_i = h_s_out - h_in  ! positive isentropic specific work of compressor (kJ/kg)
    U_tip = sqrt(1000.0_dp * w_i / psi_design)  ! rearranging definition of head coefficient and converting kJ to J
    recomp_cycle%mc%D_rotor = sqrt(m_dot / (snl_phi_design * D_in * U_tip))  ! rearranging definition of flow coefficient
    N_rad_s = U_tip * 2.0_dp / recomp_cycle%mc%D_rotor   ! shaft speed in rad/s
    recomp_cycle%mc%N_design = N_rad_s * 9.549296590_dp  ! shaft speed in rpm

    ! Set other compressor variables.
    recomp_cycle%mc%w_tip_ratio = U_tip / ssnd_out     ! ratio of the tip speed to local (comp outlet) speed of sound
    recomp_cycle%mc%eta_design = w_i / (h_out - h_in)  ! definition of isentropic efficiency
    recomp_cycle%mc%eta = recomp_cycle%mc%eta_design
    recomp_cycle%mc%phi_design = snl_phi_design
    recomp_cycle%mc%phi = snl_phi_design
    recomp_cycle%mc%phi_min = snl_phi_min
    recomp_cycle%mc%phi_max = snl_phi_max
    recomp_cycle%mc%N = recomp_cycle%mc%N_design
    recomp_cycle%mc%surge = .false.

end subroutine compressor_sizing


subroutine recompressor_sizing(recomp_cycle, error_trace)
    ! Determine the recompressor rotor diameter and design-point shaft speed
    ! and store values in recomp_cycle%rc.
    !
    ! Arguments:
    !   recomp_cycle -- a RecompCycle object that defines the simple/recompression cycle at the design point
    !   error_trace -- an ErrorTrace object

    use CO2_properties, only: CO2_TD, CO2_PS

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    type(ErrorTrace), intent(out) :: error_trace

    ! Local Variables
    integer  :: error_code
    real(dp) :: D_in, h_in, s_in, T_out, P_out, h_out, D_out, ssnd_out, h_s_out, psi_design, m_dot, w_i, U_tip, N_rad_s

    ! Create references to cycle state properties for clarity.
    D_in = recomp_cycle%dens(9)
    h_in = recomp_cycle%enth(9)
    s_in = recomp_cycle%entr(9)
    T_out = recomp_cycle%temp(10)
    P_out = recomp_cycle%pres(10)
    h_out = recomp_cycle%enth(10)
    D_out = recomp_cycle%dens(10)
    call CO2_TD(T=T_out, D=D_out, error_code=error_code, ssnd=ssnd_out)  ! speed of sound at outlet
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 132
        error_trace%files(1) = 4
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet specific enthalpy after isentropic compression
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 139
        error_trace%files(1) = 4
        return
    end if

    ! Calculate psi at the design-point phi using Horner's method
    psi_design = ((((-498626.0_dp * snl_phi_design) + 53224.0_dp) * snl_phi_design - 2505.0_dp) * snl_phi_design + 54.6_dp) &
                 * snl_phi_design + 0.04049_dp  ! from dimensionless modified head curve (at design-point, psi and modified psi are equal)

    ! Determine required size and speed of recompressor.
    m_dot = recomp_cycle%m_dot_turbine * recomp_cycle%recomp_frac  ! mass flow rate through recompressor (kg/s)
    w_i = h_s_out - h_in  ! positive isentropic specific work of recompressor (kJ/kg)
    U_tip = sqrt(1000.0_dp * w_i / psi_design)  ! rearranging definition of head coefficient and converting kJ to J
    recomp_cycle%rc%D_rotor = sqrt(m_dot / (snl_phi_design * D_in * U_tip))  ! rearranging definition of flow coefficient
    N_rad_s = U_tip * 2.0_dp / recomp_cycle%rc%D_rotor   ! shaft speed in rad/s
    recomp_cycle%rc%N_design = N_rad_s * 9.549296590_dp  ! shaft speed in rpm

    ! Set other recompressor variables.
    recomp_cycle%rc%w_tip_ratio = U_tip / ssnd_out    ! ratio of the tip speed to local (comp outlet) speed of sound
    recomp_cycle%rc%eta_design = w_i / (h_out - h_in)  ! definition of isentropic efficiency
    recomp_cycle%rc%eta = recomp_cycle%rc%eta_design
    recomp_cycle%rc%phi_design = snl_phi_design
    recomp_cycle%rc%phi = snl_phi_design
    recomp_cycle%rc%phi_min = snl_phi_min
    recomp_cycle%rc%phi_max = snl_phi_max
    recomp_cycle%rc%N = recomp_cycle%rc%N_design
    recomp_cycle%rc%surge = .false.

end subroutine recompressor_sizing
```

```
subroutine off_design_compressor(comp, T_in, P_in, m_dot, N, error_trace, T_out, P_out)
    ! Solve for the outlet state of 'comp' given its inlet conditions, mass flow rate, and shaft speed.
    !
    ! Inputs:
    !   comp -- a Compressor object, with design-point values and sizing set
    !   T_in -- compressor inlet temperature (K)
    !   P_in -- compressor inlet pressure (kPa)
    !   m_dot -- mass flow rate through compressor (kg/s)
    !   N -- shaft speed of compressor (rpm)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   T_out -- compressor outlet temperature (K)
    !   P_out -- compressor outlet pressure (kPa)
    !
    ! Notes:
    !   1) This subroutine also sets the following values in 'comp': surge, eta, w, w_tip_ratio, phi

    use CO2_Properties, only: CO2_TP, CO2_HS, CO2_PH

    ! Arguments
    type(Compressor), intent(inout) :: comp
    real(dp), intent(in) :: T_in, P_in, m_dot, N
    type(ErrorTrace), intent(out) :: error_trace
    real(dp), intent(out) :: T_out, P_out

    ! Local Variables
    integer  :: error_code
    real(dp) :: rho_in, h_in, s_in, U_tip, phi, phi_star, psi_star, eta_star, psi, eta_0, dh_s, dh, h_s_out, h_out, ssnd_out

    call CO2_TP(T=T_in, P=P_in, error_code=error_code, dens=rho_in, enth=h_in, entr=s_in)  ! fully define the inlet state of the compressor
    if (error_code /= 0) then
        error_trace%code = 1
        error_trace%lines(1) = 203
        error_trace%files(1) = 4
        return
    end if

    ! Calculate the modified flow and head coefficients and efficiency for the SNL compressor.
    U_tip = comp%D_rotor * 0.5_dp * N * 0.104719755_dp  ! tip speed in m/s
    phi = m_dot / (rho_in * U_tip * comp%D_rotor**2)    ! flow coefficient
    if (phi < comp%phi_min) then ! the compressor is operating in the surge region
        comp%surge = .true.
        phi = comp%phi_min  ! reset phi to to its minimum value; this sets psi and eta to be fixed at the values at the surge limit
    else
        comp%surge = .false.
    end if
    phi_star = phi * (N / comp%N_design)**0.2_dp  ! modified flow coefficient
    psi_star = ((((-498626.0_dp * phi_star) + 53224.0_dp) * phi_star - 2505.0_dp) * phi_star + 54.6_dp) * phi_star + 0.04049_dp  ! from ...
    eta_star = ((((-1.638e6_dp * phi_star) + 182725.0_dp) * phi_star - 8089.0_dp) * phi_star + 168.6_dp) * phi_star - 0.7069_dp  ! from ...
    psi = psi_star / ((comp%N_design / N)**((20.0_dp * phi_star)**3))
    eta_0 = eta_star * 1.47528_dp / ((comp%N_design / N)**((20.0_dp * phi_star)**5))  ! efficiency is normalized so it equals 1.0 at ...
    comp%eta = max(eta_0 * comp%eta_design, 0.0_dp)  ! the actual compressor efficiency, not allowed to go negative

    ! Check that the specified mass flow rate is possible with the compressor's current shaft speed.
    if (psi <= 0.0_dp) then  ! shaft speed is too low for the given m_dot
        error_trace%code = 1
        error_trace%lines(1) = 228
        error_trace%files(1) = 4
        return
    end if

    ! Calculate the compressor outlet state.
    dh_s = psi * U_tip**2 * 0.001_dp  ! ideal enthalpy rise in compressor, from definition of head coefficient (kJ/kg)
    dh = dh_s / comp%eta              ! actual enthalpy rise in compressor
    h_s_out = h_in + dh_s             ! ideal enthalpy at compressor outlet
    h_out = h_in + dh                 ! actual enthalpy at compressor outlet
    call CO2_HS(H=h_s_out, S=s_in, error_code=error_code, pres=P_out)  ! get the compressor outlet pressure
    if (error_code /= 0) then  ! most likely case is that the outlet pressure is above the high pressure limit of the property routine
        error_trace%code = 2
        error_trace%lines(1) = 240
        error_trace%files(1) = 4
        return
    end if
    call CO2_PH(P=P_out, H=h_out, error_code=error_code, temp=T_out, ssnd=ssnd_out)  ! determines compressor outlet temperature and speed ...
    if (error_code /= 0) then  ! most likely case is that the outlet pressure is above the high pressure limit of the property routine
        error_trace%code = 2
        error_trace%lines(1) = 247
        error_trace%files(1) = 4
        return
    end if

    ! Set a few compressor variables.
    comp%phi = phi
    comp%w_tip_ratio = U_tip / ssnd_out     ! ratio of the tip speed to local (comp outlet) speed of sound

end subroutine off_design_compressor
```

```
subroutine off_design_recompressor(comp, T_in, P_in, m_dot, P_out, error_trace, T_out)
    ! Solve for the outlet state (and shaft speed) of 'comp' given its inlet conditions, mass flow rate, and outlet pressure.
    !
    ! Inputs:
    !   comp -- a Compressor object, with design-point values and sizing set
    !   T_in -- compressor inlet temperature (K)
    !   P_in -- compressor inlet pressure (kPa)
    !   m_dot -- mass flow rate through compressor (kg/s)
    !   P_out -- compressor outlet pressure (kPa)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   T_out -- compressor outlet temperature (K)
    !
    ! Notes:
    !   1) This subroutine also sets the following values in 'comp': N, surge, eta, w, w_tip_ratio, phi
    !   2) In order to solve the compressor, the value for flow coefficient (phi) is varied until convergence.
    !   3) Surge is not allowed; if the corresponding flow coefficient is not between phi_min and phi_max an error is raised.

    use CO2_Properties, only: CO2_TP, CO2_PS, CO2_PH

    ! Arguments
    type(Compressor), intent(inout) :: comp
    real(dp), intent(in) :: T_in, P_in, m_dot, P_out
    type(ErrorTrace), intent(out) :: error_trace
    real(dp), intent(out) :: T_out

    ! Parameters
    integer, parameter :: max_iter = 100
    real(dp), parameter :: tolerance = 1.0e-9_dp  ! absolute tolerance for phi

    ! Local Variables
    integer  :: iter, error_code
    logical  :: first_pass
    real(dp) :: rho_in, h_in, s_in, alpha, phi, U_tip, phi_star, psi_star, eta_star, psi, eta_0, dh_s, dh, h_s_out, h_out, ssnd_out
    real(dp) :: N, dh_s_calc, residual, next_phi, last_phi, last_residual

    call CO2_TP(T=T_in, P=P_in, error_code=error_code, dens=rho_in, enth=h_in, entr=s_in)  ! fully define the inlet state of the compressor
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 299
        error_trace%files(1) = 4
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet enthalpy if compression/expansion is isentropic
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 306
        error_trace%files(1) = 4
        return
    end if
    dh_s = h_s_out - h_in  ! ideal enthalpy rise in compressor

    ! Iterate on phi.
    alpha = m_dot / (rho_in * comp%D_rotor**2)  ! used to reduce operation count in loop
    phi = comp%phi_design  ! start with design-point value
    first_pass = .true.
    do iter = 1, max_iter
        U_tip = alpha / phi  ! flow coefficient rearranged (with alpha substitution)
        N = (U_tip * 2.0_dp / comp%D_rotor) * 9.549296590_dp  ! shaft speed in rpm
        phi_star = phi * (N / comp%N_design)**0.2_dp  ! modified flow coefficient
        psi_star = ((((-498626.0_dp * phi_star) + 53224.0_dp) * phi_star - 2505.0_dp) * phi_star + 54.6_dp) * phi_star + 0.04049_dp  ! fro...
        psi = psi_star / ((comp%N_design / N)**((20.0_dp * phi_star)**3))
        dh_s_calc = psi * U_tip**2 * 0.001_dp  ! calculated ideal enthalpy rise in compressor, from definition of head coefficient (kJ/kg)
        residual = dh_s - dh_s_calc
        if (abs(residual) <= tolerance) exit  ! converged sufficiently
        if (first_pass) then
            next_phi = phi * 1.0001_dp  ! take a small step
            first_pass = .false.
        else
            next_phi = phi - residual * (last_phi - phi) / (last_residual - residual)  ! next guess predicted using secant method
        end if
        last_phi = phi
        last_residual = residual
        phi = next_phi
    end do

    ! Check for convergence.
    if (iter >= max_iter) then  ! did not converge
        error_trace%code = 1
        error_trace%lines(1) = 340
        error_trace%files(1) = 4
        return
    end if

    ! Calculate efficiency and outlet state.
    eta_star = ((((-1.638e6_dp * phi_star) + 182725.0_dp) * phi_star - 8089.0_dp) * phi_star + 168.6_dp) * phi_star - 0.7069_dp  ! from ...
    eta_0 = eta_star * 1.47528_dp / ((comp%N_design / N)**((20.0_dp * phi_star)**5))  ! efficiency is normalized so it equals 1.0 at ...
    comp%eta = max(eta_0 * comp%eta_design, 0.0_dp)  ! the actual compressor efficiency, not allowed to go negative
    dh = dh_s / comp%eta              ! actual enthalpy rise in compressor
    h_out = h_in + dh                 ! actual enthalpy at compressor outlet
    call CO2_PH(P=P_out, H=h_out, error_code=error_code, temp=T_out, ssnd=ssnd_out)  ! determines compressor outlet temperature and speed ...
```

```
    if (error_code /= 0) then  ! most likely case is that the outlet pressure is above the high pressure limit of the property routine
        error_trace%code = error_code
        error_trace%lines(1) = 353
        error_trace%files(1) = 4
        return
    end if
    comp%N = N
    comp%phi = phi
    comp%w_tip_ratio = U_tip / ssnd_out  ! ratio of the tip speed to local (comp outlet) speed of sound

end subroutine off_design_recompressor


end module compressors
```

snl_compressor_tsr.f90

```
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This file contains the the module 'compressors', which defines a number of subroutines based on the radial compressor being
! studied at Sandia National Laboratory.
!
! Notes:
!   1) The recompressor is modeled using two SNL compressors in series.
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: July 18, 2014
!
!-----------------------------------------------------------------------------------------------------------------------------------

module compressors

use core
implicit none
private
public :: compressor_sizing, recompressor_sizing, off_design_compressor, off_design_recompressor

real(dp), parameter :: snl_phi_design = 0.02971_dp  ! design-point flow coefficient for Sandia compressor (corresponds to max eta)
real(dp), parameter :: snl_phi_min = 0.02_dp        ! approximate surge limit for SNL compressor
real(dp), parameter :: snl_phi_max = 0.05_dp        ! approximate x-intercept for SNL compressor


contains


subroutine compressor_sizing(recomp_cycle, error_trace)
    ! Determine the compressor rotor diameter and design-point shaft speed
    ! and store values in recomp_cycle%mc.
    !
    ! Arguments:
    !   recomp_cycle -- a RecompCycle object that defines the simple/recompression cycle at the design point
    !   error_trace -- an ErrorTrace object

    use CO2_properties, only: CO2_TD, CO2_PS

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    type(ErrorTrace), intent(out) :: error_trace

    ! Local Variables
    integer  :: error_code
    real(dp) :: D_in, h_in, s_in, T_out, P_out, h_out, D_out, ssnd_out, h_s_out, psi_design, m_dot, w_i, U_tip, N_rad_s

    ! Create references to cycle state properties for clarity.
    D_in = recomp_cycle%dens(1)
    h_in = recomp_cycle%enth(1)
    s_in = recomp_cycle%entr(1)
    T_out = recomp_cycle%temp(2)
    P_out = recomp_cycle%pres(2)
    h_out = recomp_cycle%enth(2)
    D_out = recomp_cycle%dens(2)
    call CO2_TD(T=T_out, D=D_out, error_code=error_code, ssnd=ssnd_out)  ! speed of sound at outlet
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 68
        error_trace%files(1) = 4
        return
    end if
```

```
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet specific enthalpy after isentropic compression
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 75
        error_trace%files(1) = 4
        return
    end if

    ! Calculate psi at the design-point phi using Horner's method
    psi_design = ((((-498626.0_dp * snl_phi_design) + 53224.0_dp) * snl_phi_design - 2505.0_dp) * snl_phi_design + 54.6_dp) &
                 * snl_phi_design + 0.04049_dp  ! from dimensionless modified head curve (at design-point, psi and modified psi are equal)

    ! Determine required size and speed of compressor.
    m_dot = recomp_cycle%m_dot_turbine * (1.0_dp - recomp_cycle%recomp_frac)  ! mass flow rate through compressor (kg/s)
    w_i = h_s_out - h_in  ! positive isentropic specific work of compressor (kJ/kg)
    U_tip = sqrt(1000.0_dp * w_i / psi_design)  ! rearranging definition of head coefficient and converting kJ to J
    recomp_cycle%mc%D_rotor = sqrt(m_dot / (snl_phi_design * D_in * U_tip))  ! rearranging definition of flow coefficient
    N_rad_s = U_tip * 2.0_dp / recomp_cycle%mc%D_rotor   ! shaft speed in rad/s
    recomp_cycle%mc%N_design = N_rad_s * 9.549296590_dp  ! shaft speed in rpm

    ! Set other compressor variables.
    recomp_cycle%mc%w_tip_ratio = U_tip / ssnd_out     ! ratio of the tip speed to local (comp outlet) speed of sound
    recomp_cycle%mc%eta_design = w_i / (h_out - h_in)  ! definition of isentropic efficiency
    recomp_cycle%mc%eta = recomp_cycle%mc%eta_design
    recomp_cycle%mc%phi_design = snl_phi_design
    recomp_cycle%mc%phi = snl_phi_design
    recomp_cycle%mc%phi_min = snl_phi_min
    recomp_cycle%mc%phi_max = snl_phi_max
    recomp_cycle%mc%N = recomp_cycle%mc%N_design
    recomp_cycle%mc%surge = .false.

end subroutine compressor_sizing


subroutine recompressor_sizing(recomp_cycle, error_trace)
    ! Determine the recompressor rotor diameter and design-point shaft speed
    ! and store values in recomp_cycle%rc.
    !
    ! Arguments:
    !   recomp_cycle -- a RecompCycle object that defines the simple/recompression cycle at the design point
    !   error_trace -- an ErrorTrace object

    use CO2_properties, only: CO2_TD, CO2_PS, CO2_PH

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    type(ErrorTrace), intent(out) :: error_trace

    ! Parameters
    integer, parameter  :: max_iter = 100
    real(dp), parameter :: tolerance = 1.0e-8_dp  ! absolute tolerance for phi and stage efficiency

    ! Local Variables
    integer  :: iter, error_code
    real(dp) :: P_in, D_in, h_in, s_in, T_out, P_out, h_out, D_out, ssnd_out, h_s_out, psi_design, m_dot, w_i, U_tip_1, U_tip_2
    real(dp) :: eta_design, P_int, D_int, h_int, s_int, ssnd_int, lower_bound, upper_bound, N_design, D_rotor_1, D_rotor_2, w, phi
    real(dp) :: eta_stage, eta_2_req, residual, last_residual, P_secant, last_P_int, secant_step, N_rad_s

    ! Create references to cycle state properties for clarity.
    P_in = recomp_cycle%pres(9)
    D_in = recomp_cycle%dens(9)
    h_in = recomp_cycle%enth(9)
    s_in = recomp_cycle%entr(9)
    T_out = recomp_cycle%temp(10)
    P_out = recomp_cycle%pres(10)
    h_out = recomp_cycle%enth(10)
    D_out = recomp_cycle%dens(10)

    ! Set a few variables that apply to the whole recompressor.
    call CO2_TD(T=T_out, D=D_out, error_code=error_code, ssnd=ssnd_out)  ! speed of sound at outlet
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 144
        error_trace%files(1) = 4
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! ideal specific enthalpy after compression
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 151
        error_trace%files(1) = 4
        return
    end if
    eta_design = (h_s_out - h_in) / (h_out - h_in)  ! overall isentropic efficiency
    m_dot = recomp_cycle%m_dot_turbine * recomp_cycle%recomp_frac  ! mass flow rate through recompressor (kg/s)
    psi_design = ((((-498626.0_dp * snl_phi_design) + 53224.0_dp) * snl_phi_design - 2505.0_dp) * snl_phi_design + 54.6_dp) &
                 * snl_phi_design + 0.04049_dp  ! from dimensionless modified head curve (at design-point, psi and modified psi are equal)

    ! Prepare intermediate pressure iteration loop.
    last_residual = 0.0_dp
    last_P_int = 1.0e12_dp  ! ensures bisection will be used for first step
    lower_bound = P_in + 1e-6_dp
```

```
    upper_bound = P_out - 1e-6_dp
    P_int = (lower_bound + upper_bound) * 0.5_dp
    eta_stage = eta_design  ! first guess for stage efficiency
    do iter = 1, max_iter

        ! First stage
        call CO2_PS(P=P_int, S=s_in, error_code=error_code, enth=h_s_out)  ! ideal outlet specific enthalpy after first stage
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 173
            error_trace%files(1) = 4
            return
        end if
        w_i = h_s_out - h_in  ! positive isentropic specific work of first stage
        U_tip_1 = sqrt(1000.0_dp * w_i / psi_design)  ! rearranging definition of head coefficient and converting kJ to J
        D_rotor_1 = sqrt(m_dot / (snl_phi_design * D_in * U_tip_1))  ! rearranging definition of flow coefficient
        N_rad_s = U_tip_1 * 2.0_dp / D_rotor_1   ! shaft speed in rad/s
        N_design = N_rad_s * 9.549296590_dp  ! shaft speed in rpm
        w = w_i / eta_stage  ! actual first-stage work
        h_int = h_in + w  ! energy balance on first stage
        call CO2_PH(P=P_int, H=h_int, error_code=error_code, dens=D_int, entr=s_int, ssnd=ssnd_int)
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 187
            error_trace%files(1) = 4
            return
        end if

        ! Second stage
        call CO2_PS(P=P_out, S=s_int, error_code=error_code, enth=h_s_out)  ! ideal outlet specific enthalpy after second stage
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 196
            error_trace%files(1) = 4
            return
        end if
        w_i = h_s_out - h_int  ! positive isentropic specific work of second stage
        U_tip_2 = sqrt(1000.0_dp * w_i / psi_design)  ! rearranging definition of head coefficient and converting kJ to J
        D_rotor_2 = 2.0_dp * U_tip_2 / (N_design * 0.104719755_dp)  ! required second-stage diameter
        phi = m_dot / (D_int * U_tip_2 * D_rotor_2**2)  ! required flow coefficient
        eta_2_req = w_i / (h_out - h_int)  ! required second stage efficiency to achieve overall eta_design

        ! Check convergence and update guesses.
        residual = snl_phi_design - phi
        if (residual < 0.0_dp) then  ! P_int guess is too high
            if (-residual <= tolerance .and. abs(eta_stage-eta_2_req) <= tolerance) exit
            upper_bound = P_int
        else  ! P_int guess is too low
            if (residual <= tolerance .and. abs(eta_stage-eta_2_req) <= tolerance) exit
            lower_bound = P_int
        end if
        secant_step = -residual * (last_P_int - P_int) / (last_residual - residual)
        P_secant = P_int + secant_step
        last_P_int = P_int
        last_residual = residual
        if (P_secant <= lower_bound .or. P_secant >= upper_bound) then  ! secant method overshot
            P_int = (lower_bound + upper_bound) * 0.5_dp
        else if (abs(secant_step) > abs((upper_bound - lower_bound) * 0.5_dp)) then  ! take the smaller step to ensure convergence
            P_int = (lower_bound + upper_bound) * 0.5_dp
        else
            P_int = P_secant  ! use secant guess
        end if
        eta_stage = 0.5_dp * (eta_stage + eta_2_req)  ! update guess for stage efficiency
    end do

    ! Check for convergence.
    if (iter >= max_iter) then  ! did not converge
        error_trace%code = 1
        error_trace%lines(1) = 233
        error_trace%files(1) = 4
        return
    end if

    ! Set recompressor variables.
    recomp_cycle%rc%D_rotor = D_rotor_1
    recomp_cycle%rc%D_rotor_2 = D_rotor_2
    recomp_cycle%rc%N_design = N_design
    recomp_cycle%rc%eta_design = eta_stage
    recomp_cycle%rc%phi_design = snl_phi_design
    recomp_cycle%rc%phi_min = snl_phi_min
    recomp_cycle%rc%phi_max = snl_phi_max
    recomp_cycle%rc%N = N_design
    recomp_cycle%rc%eta = eta_design
    recomp_cycle%rc%phi = snl_phi_design
    recomp_cycle%rc%phi_2 = snl_phi_design
    recomp_cycle%rc%w_tip_ratio = max(U_tip_1 / ssnd_int, U_tip_2 / ssnd_out)
    recomp_cycle%rc%surge = .false.

end subroutine recompressor_sizing
```

```
subroutine off_design_compressor(comp, T_in, P_in, m_dot, N, error_trace, T_out, P_out)
    ! Solve for the outlet state of 'comp' given its inlet conditions, mass flow rate, and shaft speed.
    !
    ! Inputs:
    !   comp -- a Compressor object, with design-point values and sizing set
    !   T_in -- compressor inlet temperature (K)
    !   P_in -- compressor inlet pressure (kPa)
    !   m_dot -- mass flow rate through compressor (kg/s)
    !   N -- shaft speed of compressor (rpm)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   T_out -- compressor outlet temperature (K)
    !   P_out -- compressor outlet pressure (kPa)
    !
    ! Notes:
    !   1) This subroutine also sets the following values in 'comp': surge, eta, w, w_tip_ratio, phi

    use CO2_Properties, only: CO2_TP, CO2_HS, CO2_PH

    ! Arguments
    type(Compressor), intent(inout) :: comp
    real(dp), intent(in) :: T_in, P_in, m_dot, N
    type(ErrorTrace), intent(out) :: error_trace
    real(dp), intent(out) :: T_out, P_out

    ! Local Variables
    integer  :: error_code
    real(dp) :: rho_in, h_in, s_in, U_tip, phi, phi_star, psi_star, eta_star, psi, eta_0, dh_s, dh, h_s_out, h_out, ssnd_out

    call CO2_TP(T=T_in, P=P_in, error_code=error_code, dens=rho_in, enth=h_in, entr=s_in)  ! fully define the inlet state of the compressor
    if (error_code /= 0) then
        error_trace%code = 1
        error_trace%lines(1) = 288
        error_trace%files(1) = 4
        return
    end if

    ! Calculate the modified flow and head coefficients and efficiency for the SNL compressor.
    U_tip = comp%D_rotor * 0.5_dp * N * 0.104719755_dp  ! tip speed in m/s
    phi = m_dot / (rho_in * U_tip * comp%D_rotor**2)    ! flow coefficient
    if (phi < comp%phi_min) then ! the compressor is operating in the surge region
        comp%surge = .true.
        phi = comp%phi_min  ! reset phi to to its minimum value; this sets psi and eta to be fixed at the values at the surge limit
    else
        comp%surge = .false.
    end if
    phi_star = phi * (N / comp%N_design)**0.2_dp  ! modified flow coefficient
    psi_star = ((((-498626.0_dp * phi_star) + 53224.0_dp) * phi_star - 2505.0_dp) * phi_star + 54.6_dp) * phi_star + 0.04049_dp  ! from ...
    eta_star = ((((-1.638e6_dp * phi_star) + 182725.0_dp) * phi_star - 8089.0_dp) * phi_star + 168.6_dp) * phi_star - 0.7069_dp  ! from ...
    psi = psi_star / ((comp%N_design / N)**((20.0_dp * phi_star)**3))
    eta_0 = eta_star * 1.47528_dp / ((comp%N_design / N)**((20.0_dp * phi_star)**5))  ! efficiency is normalized so it equals 1.0 at ...
    comp%eta = max(eta_0 * comp%eta_design, 0.0_dp)  ! the actual compressor efficiency, not allowed to go negative

    ! Check that the specified mass flow rate is possible with the compressor's current shaft speed.
    if (psi <= 0.0_dp) then  ! shaft speed is too low for the given m_dot
        error_trace%code = 1
        error_trace%lines(1) = 313
        error_trace%files(1) = 4
        return
    end if

    ! Calculate the compressor outlet state.
    dh_s = psi * U_tip**2 * 0.001_dp  ! ideal enthalpy rise in compressor, from definition of head coefficient (kJ/kg)
    dh = dh_s / comp%eta              ! actual enthalpy rise in compressor
    h_s_out = h_in + dh_s             ! ideal enthalpy at compressor outlet
    h_out = h_in + dh                 ! actual enthalpy at compressor outlet
    call CO2_HS(H=h_s_out, S=s_in, error_code=error_code, pres=P_out)  ! get the compressor outlet pressure
    if (error_code /= 0) then  ! most likely case is that the outlet pressure is above the high pressure limit of the property routine
        error_trace%code = 2
        error_trace%lines(1) = 325
        error_trace%files(1) = 4
        return
    end if
    call CO2_PH(P=P_out, H=h_out, error_code=error_code, temp=T_out, ssnd=ssnd_out)  ! determines compressor outlet temperature and speed ...
    if (error_code /= 0) then  ! most likely case is that the outlet pressure is above the high pressure limit of the property routine
        error_trace%code = 2
        error_trace%lines(1) = 332
        error_trace%files(1) = 4
        return
    end if

    ! Set a few compressor variables.
    comp%phi = phi
    comp%w_tip_ratio = U_tip / ssnd_out     ! ratio of the tip speed to local (comp outlet) speed of sound

end subroutine off_design_compressor
```

```
subroutine off_design_recompressor(comp, T_in, P_in, m_dot, P_out, error_trace, T_out)
    ! Solve for the outlet state (and shaft speed) of 'comp' given its inlet conditions, mass flow rate, and outlet pressure.
    !
    ! Inputs:
    !   comp -- a Compressor object, with design-point values and sizing set
    !   T_in -- compressor inlet temperature (K)
    !   P_in -- compressor inlet pressure (kPa)
    !   m_dot -- mass flow rate through compressor (kg/s)
    !   P_out -- compressor outlet pressure (kPa)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   T_out -- compressor outlet temperature (K)
    !
    ! Notes:
    !   1) This subroutine also sets the following values in 'comp': N, surge, eta, w, w_tip_ratio, phi
    !   2) In order to solve the compressor, the value for flow coefficient (phi) is varied until convergence.
    !   3) Surge is not allowed; if the corresponding flow coefficient is not between phi_min and phi_max an error is raised.
    !   4) Two-stage recompressor; surge is true if either stages are in surge conditions; w_tip_ratio is max of the two stages.

    use CO2_Properties, only: CO2_TP, CO2_PS, CO2_PH, CO2_HS

    ! Arguments
    type(Compressor), intent(inout) :: comp
    real(dp), intent(in) :: T_in, P_in, m_dot, P_out
    type(ErrorTrace), intent(out) :: error_trace
    real(dp), intent(out) :: T_out

    ! Parameters
    integer, parameter :: max_iter = 100
    real(dp), parameter :: rel_tol = 1.0e-9_dp  ! relative tolerance for pressure

    ! Local Variables
    integer  :: iter, error_code
    logical  :: first_pass
    real(dp) :: rho_in, h_in, s_in, phi_1, U_tip_1, phi_star, psi_star, eta_star, psi, eta_0, dh_s, dh, h_s_out, h_out, ssnd_out
    real(dp) :: N, dh_s_calc, residual, next_phi, last_phi_1, last_residual, P_int, h_int, D_int, s_int, ssnd_int, eta_stage_1
    real(dp) :: eta_stage_2, phi_2, U_tip_2, P_out_calc

    call CO2_TP(T=T_in, P=P_in, error_code=error_code, dens=rho_in, enth=h_in, entr=s_in)  ! fully define the inlet state of the compressor
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 386
        error_trace%files(1) = 4
        return
    end if

    ! Iterate on first-stage phi.
    phi_1 = comp%phi_design  ! start with design-point value
    first_pass = .true.
    do iter = 1, max_iter

        ! First stage - dh_s and eta_stage_1.
        U_tip_1 = m_dot / (phi_1 * rho_in * comp%D_rotor**2)  ! flow coefficient rearranged
        N = (U_tip_1 * 2.0_dp / comp%D_rotor) * 9.549296590_dp  ! shaft speed in rpm
        phi_star = phi_1 * (N / comp%N_design)**0.2_dp  ! modified flow coefficient
        psi_star = ((((-498626.0_dp * phi_star) + 53224.0_dp) * phi_star - 2505.0_dp) * phi_star + 54.6_dp) * phi_star + 0.04049_dp  ! fro...
        psi = psi_star / ((comp%N_design / N)**((20.0_dp * phi_star)**3))
        dh_s = psi * U_tip_1**2 * 0.001_dp  ! calculated ideal enthalpy rise in first stage of compressor, from definition of head ...
        eta_star = ((((-1.638e6_dp * phi_star) + 182725.0_dp) * phi_star - 8089.0_dp) * phi_star + 168.6_dp) * phi_star - 0.7069_dp  ! fro...
        eta_0 = eta_star * 1.47528_dp / ((comp%N_design / N)**((20.0_dp * phi_star)**5))  ! stage efficiency is normalized so it equals ...
        eta_stage_1 = max(eta_0 * comp%eta_design, 0.0_dp)  ! the actual stage efficiency, not allowed to go negative

        ! Calculate first-stage outlet (second-stage inlet) state.
        dh = dh_s / eta_stage_1             ! actual enthalpy rise in first stage
        h_s_out = h_in + dh_s              ! ideal enthalpy between stages
        h_int = h_in + dh                  ! actual enthalpy between stages
        call CO2_HS(H=h_s_out, S=s_in, error_code=error_code, pres=P_int)  ! get the first-stage outlet pressure (second-stage inlet ...
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 414
            error_trace%files(1) = 4
            return
        end if
        call CO2_PH(P=P_int, H=h_int, error_code=error_code, dens=D_int, entr=s_int, ssnd=ssnd_int)  ! get second-stage inlet properties
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 421
            error_trace%files(1) = 4
            return
        end if

        ! Second stage - dh_s and eta_stage_2.
        U_tip_2 = comp%D_rotor_2 * 0.5_dp * N * 0.104719755_dp  ! second-stage tip speed in m/s
        phi_2 = m_dot / (D_int * U_tip_2 * comp%D_rotor_2**2)  ! second-stage flow coefficient
        phi_star = phi_2 * (N / comp%N_design)**0.2_dp  ! modified flow coefficient
        psi_star = ((((-498626.0_dp * phi_star) + 53224.0_dp) * phi_star - 2505.0_dp) * phi_star + 54.6_dp) * phi_star + 0.04049_dp  ! fro...
        psi = psi_star / ((comp%N_design / N)**((20.0_dp * phi_star)**3))
        dh_s = psi * U_tip_2**2 * 0.001_dp  ! calculated ideal enthalpy rise in second stage of compressor, from definition of head ...
        eta_star = ((((-1.638e6_dp * phi_star) + 182725.0_dp) * phi_star - 8089.0_dp) * phi_star + 168.6_dp) * phi_star - 0.7069_dp  ! fro...
        eta_0 = eta_star * 1.47528_dp / ((comp%N_design / N)**((20.0_dp * phi_star)**5))  ! stage efficiency is normalized so it equals ...
        eta_stage_2 = max(eta_0 * comp%eta_design, 0.0_dp)  ! the actual stage efficiency, not allowed to go negative
```

```
        ! Calculate second-stage outlet state.
        dh = dh_s / eta_stage_2            ! actual enthalpy rise in second stage
        h_s_out = h_int + dh_s             ! ideal enthalpy at compressor outlet
        h_out = h_int + dh                 ! actual enthalpy at compressor outlet
        call CO2_HS(H=h_s_out, S=s_int, error_code=error_code, pres=P_out_calc)  ! get the calculated compressor outlet pressure
        if (error_code /= 0) then
            error_trace%code = error_code
            error_trace%lines(1) = 444
            error_trace%files(1) = 4
            return
        end if

        ! Check for convergence and adjust phi_1 guess.
        residual = P_out - P_out_calc
        if (abs(residual) / P_out <= rel_tol) exit  ! converged sufficiently
        if (first_pass) then
            next_phi = phi_1 * 1.0001_dp  ! take a small step
            first_pass = .false.
        else
            next_phi = phi_1 - residual * (last_phi_1 - phi_1) / (last_residual - residual)  ! next guess predicted using secant method
        end if
        last_phi_1 = phi_1
        last_residual = residual
        phi_1 = next_phi

    end do

    ! Check for convergence.
    if (iter >= max_iter) then  ! did not converge
        error_trace%code = 1
        error_trace%lines(1) = 468
        error_trace%files(1) = 4
        return
    end if

    ! Determine outlet temperature and speed of sound.
    call CO2_PH(P=P_out_calc, H=h_out, error_code=error_code, temp=T_out, ssnd=ssnd_out)
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 476
        error_trace%files(1) = 4
        return
    end if
    call CO2_PS(P=P_out_calc, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet specific enthalpy after isentropic compression
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 483
        error_trace%files(1) = 4
        return
    end if

    ! Set relevant recompressor variables.
    comp%N = N
    comp%eta = (h_s_out - h_in) / (h_out - h_in)  ! use overall isentropic efficiency
    comp%phi = phi_1
    comp%phi_2 = phi_2
    comp%w_tip_ratio = max(U_tip_1 / ssnd_int, U_tip_2 / ssnd_out)  ! store maximum ratio
    comp%surge = (phi_1 < comp%phi_min .or. phi_2 < comp%phi_min)

end subroutine off_design_recompressor


end module compressors
```

Appendix V: `turbines` Module

radial_turbine.f90

```
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This file contains the the module 'turbines', defining a number of subroutines based on a generic, low-reaction radial turbine.
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: August 14, 2014
!
!-----------------------------------------------------------------------------------------------------------------------------------

module turbines

use core
implicit none
private
public :: turbine_sizing, off_design_turbine

real(dp), parameter :: nu_design = 0.707_dp  ! design-point ratio of tip speed to spouting velocity at maximum efficiency


contains


subroutine turbine_sizing(recomp_cycle, error_trace)
    ! Determine the turbine rotor diameter, effective nozzle area, and design-point shaft
    ! speed and store values in recomp_cycle%t.
    !
    ! Arguments:
    !   recomp_cycle -- a RecompCycle object that defines the simple/recompression cycle at the design point
    !   error_trace -- an ErrorTrace object
    !
    ! Notes:
    !   1) The value for recomp_cycle%t%N_design is required to be set.  If it is <= 0.0 then
    !      the value for recomp_cycle%mc%N_design is used (i.e., link the compressor and turbine
    !      shafts).  For this reason, turbine_sizing must be called after compressor_sizing if
    !      the shafts are to be linked.

    use CO2_properties, only: CO2_TD, CO2_PS

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    type(ErrorTrace), intent(out) :: error_trace

    ! Local Variables
    integer :: error_code
    real(dp) :: T_in, D_in, h_in, s_in, P_out, h_out, D_out, ssnd_in, h_s_out, w_i, C_s, U_tip

    ! Check that a design-point shaft speed is available.
    if (recomp_cycle%t%N_design <= 0.0_dp) then  ! link shafts
        recomp_cycle%t%N_design = recomp_cycle%mc%N_design
        if (recomp_cycle%mc%N_design <= 0.0_dp) then
            error_trace%code = 7
            error_trace%lines(1) = 61
            error_trace%files(1) = 5
            return
        end if
    end if

    ! Create references to cycle state properties for clarity.
    T_in = recomp_cycle%temp(6)
    D_in = recomp_cycle%dens(6)
    h_in = recomp_cycle%enth(6)
    s_in = recomp_cycle%entr(6)
    P_out = recomp_cycle%pres(7)
    h_out = recomp_cycle%enth(7)
    D_out = recomp_cycle%dens(7)
    call CO2_TD(T=T_in, D=D_in, error_code=error_code, ssnd=ssnd_in)  ! speed of sound at inlet
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 79
```

```
        error_trace%files(1) = 5
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet specific enthalpy after isentropic expansion
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 86
        error_trace%files(1) = 5
        return
    end if

    ! Determine necessary turbine parameters.
    recomp_cycle%t%nu = nu_design
    w_i = h_in - h_s_out  ! isentropic specific work of turbine (kJ/kg)
    C_s = sqrt(2.0_dp * w_i * 1000.0_dp)  ! spouting velocity in m/s
    U_tip = recomp_cycle%t%nu * C_s  ! rearrange definition of nu
    recomp_cycle%t%D_rotor = U_tip / (0.5_dp * recomp_cycle%t%N_design * 0.104719755_dp)  ! turbine diameter in m
    recomp_cycle%t%A_nozzle = recomp_cycle%m_dot_turbine / (C_s * D_out)  ! turbine effective nozzle area in m2

    ! Set other turbine variables.
    recomp_cycle%t%w_tip_ratio = U_tip / ssnd_in  ! ratio of the tip speed to local (turbine inlet) speed of sound
    recomp_cycle%t%eta_design = (h_in - h_out) / w_i  ! definition of isentropic efficiency
    recomp_cycle%t%eta = recomp_cycle%t%eta_design
    recomp_cycle%t%N = recomp_cycle%t%N_design

end subroutine turbine_sizing


subroutine off_design_turbine(turb, T_in, P_in, P_out, N, error_trace, m_dot, T_out)
    ! Solve for the outlet state of 'turb' given its inlet conditions, outlet pressure, and shaft speed.
    !
    ! Inputs:
    !   turb -- a Turbine object, with design-point values and sizing set
    !   T_in -- turbine inlet temperature (K)
    !   P_in -- turbine inlet pressure (kPa)
    !   P_out -- turbine outlet pressure (kPa)
    !   N -- shaft speed of turbine (rpm)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   m_dot -- allowable mass flow rate through the turbine (kg/s)
    !   T_out -- turbine outlet temperature (K)
    !
    ! Notes:
    !   1) This subroutine also sets the following values in 'turb': nu, eta, m_dot, w, w_tip_ratio

    use CO2_Properties, only: CO2_TP, CO2_PS, CO2_PH

    ! Arguments
    type(Turbine), intent(inout) :: turb
    real(dp), intent(in) :: T_in, P_in, P_out, N
    type(ErrorTrace), intent(out) :: error_trace
    real(dp), intent(out) :: m_dot, T_out

    ! Local Variables
    integer  :: error_code
    real(dp) :: h_in, s_in, ssnd_in, U_tip, h_s_out, h_out, D_out, C_s, eta_0

    call CO2_TP(T=T_in, P=P_in, error_code=error_code, enth=h_in, entr=s_in, ssnd=ssnd_in)  ! properties at inlet of turbine
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 141
        error_trace%files(1) = 5
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! enthalpy at the turbine outlet if the expansion is isentropic
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 148
        error_trace%files(1) = 5
        return
    end if

    ! Apply the radial turbine equations for efficiency.
    C_s = sqrt(2.0_dp * (h_in - h_s_out) * 1000.0_dp)  ! spouting velocity (m/s)
    U_tip = turb%D_rotor * 0.5_dp * N * 0.104719755_dp  ! turbine tip speed (m/s)
    turb%nu = U_tip / C_s  ! ratio of tip speed to spouting velocity
    if (turb%nu < 1.0_dp) then
        eta_0 = 2.0_dp * turb%nu * sqrt(1.0_dp - turb%nu**2)  ! efficiency from Baines (1.0 at design point)
    else
        eta_0 = 0.0_dp  ! catches nu values just over 1, which leads to sqrt of negative number
    end if
    turb%eta = eta_0 * turb%eta_design  ! actual turbine efficiency

    ! Calculate the outlet state and allowable mass flow rate.
    h_out = h_in - turb%eta * (h_in - h_s_out)  ! enthalpy at turbine outlet
    call CO2_PH(P=P_out, H=h_out, error_code=error_code, temp=T_out, dens=D_out)
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 169
        error_trace%files(1) = 5
        return
```

```
    end if
    m_dot = C_s * turb%A_nozzle * D_out  ! mass flow through turbine (kg/s)
    turb%w_tip_ratio = U_tip / ssnd_in  ! ratio of the tip speed to the local (turbine inlet) speed of sound
    turb%N = N

end subroutine off_design_turbine


end module turbines
```

snl_radial_turbine.f90

```
!-----------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-----------------------------------------------------------------------------------------------------------------------------
!
! This file contains the the module 'turbines', defining a number of subroutines based on the SNL radial turbine.
!
! Notes:
!   1) This model is very similar to the model defined in radial_turbine.f90, with the exception of a modified efficiency
!      curve (and hence design-point nu value) and the use of the compressor inlet density in the allowable mass flow
!      rate equation (as opposed to the outlet density in the low-reaction radial turbine model).
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: August 20, 2014
!
!-----------------------------------------------------------------------------------------------------------------------------

module turbines

use core
implicit none
private
public :: turbine_sizing, off_design_turbine

real(dp), parameter :: nu_design = 0.7476_dp  ! maximizes efficiency for SNL turbine efficiency curve


contains


subroutine turbine_sizing(recomp_cycle, error_trace)
    ! Determine the turbine rotor diameter, effective nozzle area, and design-point shaft
    ! speed and store values in recomp_cycle%t.
    !
    ! Arguments:
    !   recomp_cycle -- a RecompCycle object that defines the simple/recompression cycle at the design point
    !   error_trace -- an ErrorTrace object
    !
    ! Notes:
    !   1) The value for recomp_cycle%t%N_design is required to be set.  If it is <= 0.0 then
    !      the value for recomp_cycle%mc%N_design is used (i.e., link the compressor and turbine
    !      shafts).  For this reason, turbine_sizing must be called after compressor_sizing if
    !      the shafts are to be linked.

    use CO2_properties, only: CO2_TD, CO2_PS

    ! Arguments
    type(RecompCycle), intent(inout) :: recomp_cycle
    type(ErrorTrace), intent(out) :: error_trace

    ! Local Variables
    integer :: error_code
    real(dp) :: T_in, D_in, h_in, s_in, P_out, h_out, D_out, ssnd_in, h_s_out, w_i, C_s, U_tip

    ! Check that a design-point shaft speed is available.
    if (recomp_cycle%t%N_design <= 0.0_dp) then  ! link shafts
        recomp_cycle%t%N_design = recomp_cycle%mc%N_design
        if (recomp_cycle%mc%N_design <= 0.0_dp) then
            error_trace%code = 7
            error_trace%lines(1) = 65
            error_trace%files(1) = 5
            return
        end if
    end if

    ! Create references to cycle state properties for clarity.
    T_in = recomp_cycle%temp(6)
    D_in = recomp_cycle%dens(6)
    h_in = recomp_cycle%enth(6)
```

```
    s_in = recomp_cycle%entr(6)
    P_out = recomp_cycle%pres(7)
    h_out = recomp_cycle%enth(7)
    D_out = recomp_cycle%dens(7)
    call CO2_TD(T=T_in, D=D_in, error_code=error_code, ssnd=ssnd_in)  ! speed of sound at inlet
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 84
        error_trace%files(1) = 5
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! outlet specific enthalpy after isentropic expansion
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 91
        error_trace%files(1) = 5
        return
    end if

    ! Determine necessary turbine parameters.
    recomp_cycle%t%nu = nu_design
    w_i = h_in - h_s_out  ! isentropic specific work of turbine (kJ/kg)
    C_s = sqrt(2.0_dp * w_i * 1000.0_dp)  ! spouting velocity in m/s
    U_tip = recomp_cycle%t%nu * C_s  ! rearrange definition of nu
    recomp_cycle%t%D_rotor = U_tip / (0.5_dp * recomp_cycle%t%N_design * 0.104719755_dp)  ! turbine diameter in m
    recomp_cycle%t%A_nozzle = recomp_cycle%m_dot_turbine / (C_s * D_in)  ! turbine effective nozzle area in m2

    ! Set other turbine variables.
    recomp_cycle%t%w_tip_ratio = U_tip / ssnd_in  ! ratio of the tip speed to local (turbine inlet) speed of sound
    recomp_cycle%t%eta_design = (h_in - h_out) / w_i  ! definition of isentropic efficiency
    recomp_cycle%t%eta = recomp_cycle%t%eta_design
    recomp_cycle%t%N = recomp_cycle%t%N_design

end subroutine turbine_sizing


subroutine off_design_turbine(turb, T_in, P_in, P_out, N, error_trace, m_dot, T_out)
    ! Solve for the outlet state of 'turb' given its inlet conditions, outlet pressure, and shaft speed.
    !
    ! Inputs:
    !   turb -- a Turbine object, with design-point values and sizing set
    !   T_in -- turbine inlet temperature (K)
    !   P_in -- turbine inlet pressure (kPa)
    !   P_out -- turbine outlet pressure (kPa)
    !   N -- shaft speed of turbine (rpm)
    !
    ! Outputs:
    !   error_trace -- an ErrorTrace object
    !   m_dot -- allowable mass flow rate through the turbine (kg/s)
    !   T_out -- turbine outlet temperature (K)
    !
    ! Notes:
    !   1) This subroutine also sets the following values in 'turb': nu, eta, m_dot, w, w_tip_ratio

    use CO2_Properties, only: CO2_TP, CO2_PS, CO2_PH

    ! Arguments
    type(Turbine), intent(inout) :: turb
    real(dp), intent(in) :: T_in, P_in, P_out, N
    type(ErrorTrace), intent(out) :: error_trace
    real(dp), intent(out) :: m_dot, T_out

    ! Local Variables
    integer  :: error_code
    real(dp) :: h_in, s_in, ssnd_in, U_tip, h_s_out, h_out, D_in, D_out, C_s, eta_0

    call CO2_TP(T=T_in, P=P_in, error_code=error_code, dens=D_in, enth=h_in, entr=s_in, ssnd=ssnd_in)  ! properties at inlet of turbine
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 146
        error_trace%files(1) = 5
        return
    end if
    call CO2_PS(P=P_out, S=s_in, error_code=error_code, enth=h_s_out)  ! enthalpy at the turbine outlet if the expansion is isentropic
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 153
        error_trace%files(1) = 5
        return
    end if

    ! Apply the radial turbine equations for efficiency.
    C_s = sqrt(2.0_dp * (h_in - h_s_out) * 1000.0_dp)  ! spouting velocity (m/s)
    U_tip = turb%D_rotor * 0.5_dp * N * 0.104719755_dp  ! turbine tip speed (m/s)
    turb%nu = U_tip / C_s  ! ratio of tip speed to spouting velocity

    !eta_0 = 0.179921180_dp + 1.3567_dp*turb%nu + 1.3668_dp*turb%nu**2 - 3.0874_dp*turb%nu**3 + 1.0626_dp*turb%nu**4
    eta_0 = (((1.0626_dp * turb%nu - 3.0874_dp) * turb%nu + 1.3668_dp) * turb%nu + 1.3567_dp) * turb%nu + 0.179921180_dp
    eta_0 = max(eta_0, 0.0_dp)
    eta_0 = min(eta_0, 1.0_dp)
    turb%eta = eta_0 * turb%eta_design  ! actual turbine efficiency
```

```
    ! Calculate the outlet state and allowable mass flow rate.
    h_out = h_in - turb%eta * (h_in - h_s_out)  ! enthalpy at turbine outlet
    call CO2_PH(P=P_out, H=h_out, error_code=error_code, temp=T_out, dens=D_out)
    if (error_code /= 0) then
        error_trace%code = error_code
        error_trace%lines(1) = 174
        error_trace%files(1) = 5
        return
    end if
    m_dot = C_s * turb%A_nozzle * D_in  ! mass flow through turbine (kg/s)
    turb%w_tip_ratio = U_tip / ssnd_in  ! ratio of the tip speed to the local (turbine inlet) speed of sound
    turb%N = N

end subroutine off_design_turbine


end module turbines
```

Appendix VI: heat_exchangers Module

scaling_hxr.f90

```
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This file contains the the module 'heat_exchangers', which defines heat exchanger pressure drop and conductance scaling functions.
!
! Notes:
!   1) Pressure drops are scaled with mass flow rate according to the Darcy friction factor and Blasius correlation.
!   2) Conductance values are scaled with mass flow rate according to the Dittus-Boelter heat transfer correlation.
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: July 12, 2014
!
!-----------------------------------------------------------------------------------------------------------------------------------

module heat_exchangers

use core
implicit none
private
public :: hxr_pressure_drops, hxr_conductance

contains

function hxr_pressure_drops(hxr, m_dots)
    ! Return an array of the scaled pressure drops (in kPa) for the two streams of the heat exchanger defined by 'hxr'.
    !
    ! Inputs:
    !   hxr -- a HeatExchanger type with design-point values set
    !   m_dots -- mass flow rates of the two streams (kg/s) [1: cold, 2: hot]
    !
    type(HeatExchanger), intent(in) :: hxr
    real(dp), dimension(2), intent(in) :: m_dots
    real(dp), dimension(2) :: hxr_pressure_drops
    hxr_pressure_drops = hxr%DP_design * (m_dots / hxr%m_dot_design)**1.75_dp  ! operates on both streams simultaneously
end function hxr_pressure_drops

real(dp) function hxr_conductance(hxr, m_dots)
    ! Return the scaled conductance (in kW/K) of the heat exchanger defined by 'hxr'.
    !
    ! Inputs:
    !   hxr -- a HeatExchanger type with design-point values set
    !   m_dots -- mass flow rates of the two streams (kg/s) [1: cold, 2: hot]
    !
    type(HeatExchanger), intent(in) :: hxr
    real(dp), dimension(2), intent(in) :: m_dots
    real(dp) :: m_dot_ratio
    m_dot_ratio = (m_dots(1) / hxr%m_dot_design(1) + m_dots(2) / hxr%m_dot_design(2)) * 0.5_dp  ! average the two streams
    hxr_conductance = hxr%UA_design * m_dot_ratio**0.8_dp
end function hxr_conductance

end module heat_exchangers
```

Appendix VII: CO2_properties Module

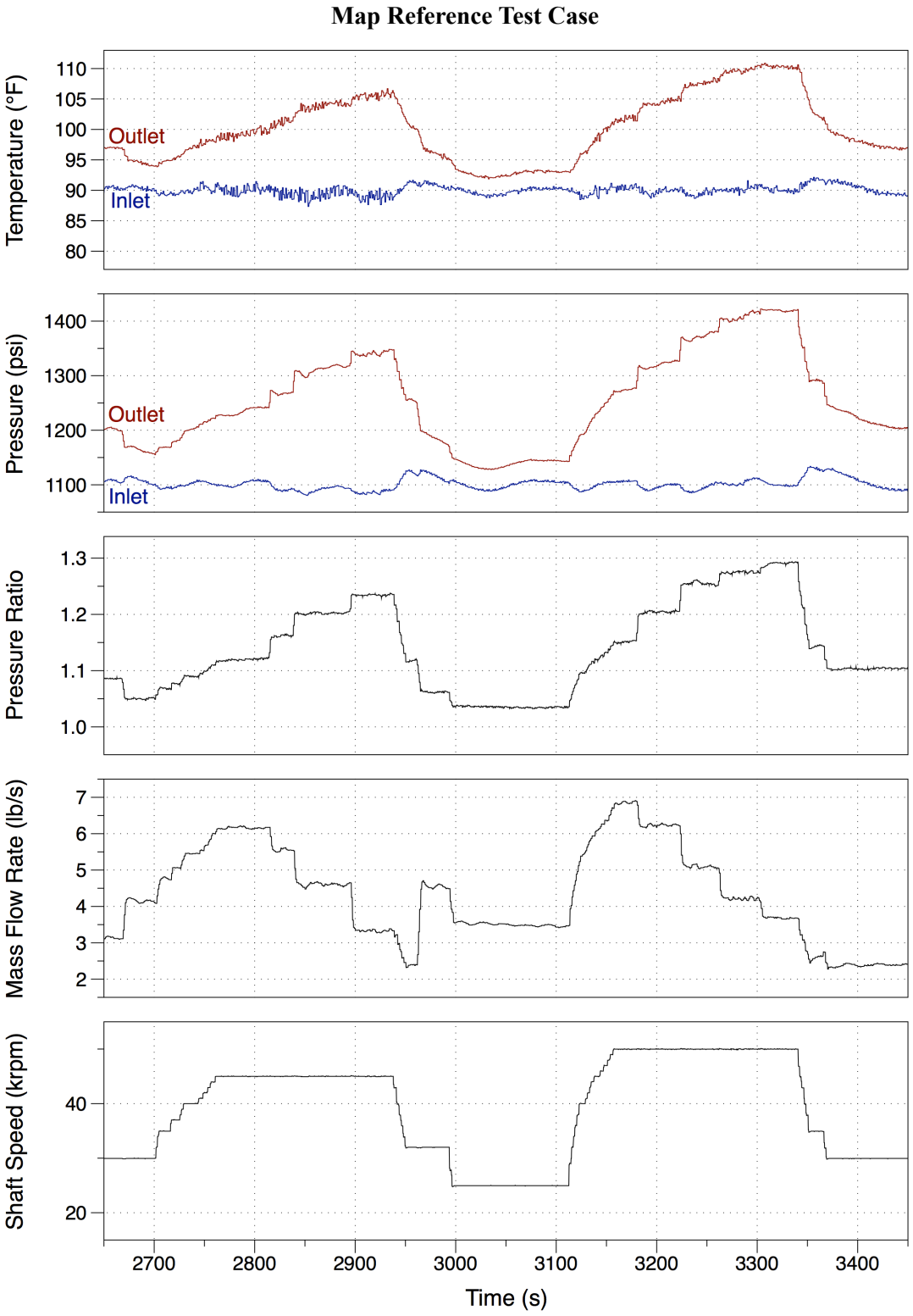
module_CO2_properties.f90 (not listed, refer to version online)

CO2_RP_module.f90

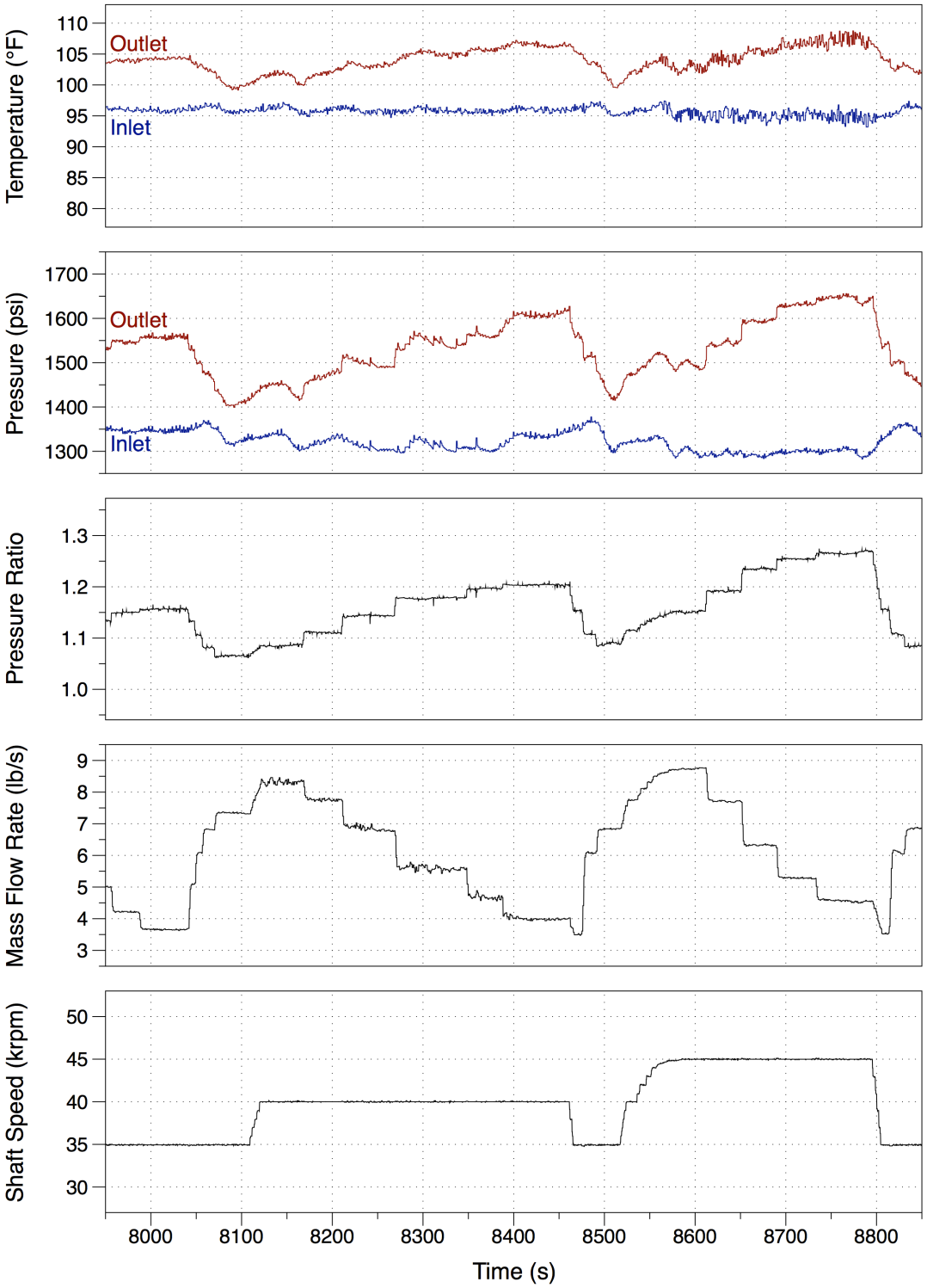
```
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This is free and unencumbered software released into the public domain.
!
! Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a
! compiled binary, for any purpose, commercial or non-commercial, and by any means.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
! MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM,
! DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
! SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
!
!-----------------------------------------------------------------------------------------------------------------------------------
!
! This file contains a module with multiple subroutines that call the REFPROP flash routines for carbon dioxide.
!
! For each subroutine, the first two arguments are the known properties:
!   T -- temperature (K)
!   D -- density (kg/m3)
!   P -- pressure (kPa)
!   H -- enthalpy (kJ/kg)
!   S -- entropy (kJ/kg-K)
!
! All the outputs except 'error_code' are optional:
!   temp -- temperature (K)
!   pres -- pressure (kPa)
!   dens -- density (kg/m3)
!   enth -- enthalpy (kJ/kg)
!   entr -- entropy (kJ/kg-K)
!   ssnd -- speed of sound in the fluid (m/s)
!
! Notes:
!   1) The REFPROP source code is not provided and must be purchased from http://www.nist.gov/srd/nist23.cfm and
!      linked to during compilation of this module.  To use REFPROP with the program "generate_paper_results.py", the
!      easiest way to do this is to copy all the REFPROP source code files into the directory containing this file; the
!      "create_python_interface.py" program will then find and compile them automatically.
!   2) The parameter 'fluid' is the full path to the CO2.FLD fluid definition file (syntax is platform specific).
!   3) If an error occurs during initialization, the calling program will be stopped.
!
! Author: John Dyreby, Solar Energy Laboratory, University of Wisconsin-Madison <jjdyreby@uwalumni.com>
!
! Last Modified: July 10, 2014
!
!-----------------------------------------------------------------------------------------------------------------------------------


module CO2_properties

implicit none

! Parameters
integer,  parameter :: nc = 1                               ! number of components in the mixture
integer, parameter  :: dp = selected_real_kind(15)          ! double precision
real(dp), parameter :: comp_array = 1.0_dp                  ! composition of the mixture
character(len=3),   parameter :: reference_state = 'DEF'    ! use the default reference state for each fluid
character(len=255), parameter :: mixture_file = 'HMX.BNC'   ! default mixture coefficients
character(len=255), parameter :: fluid = '/Path/To/REFPROP/Fluids/CO2.FLD'  ! path to CO2.FLD

! Module Variables
logical, save :: initialized = .false.
character(len=255) :: error_message
integer :: error_code
real(dp) :: wmm

contains

subroutine initialize()
    real(dp), external :: wmol
    call setup(nc, fluid, mixture_file, reference_state, error_code, error_message)
    if (error_code /= 0) then
        write (*,*) 'The following error occurred during REFPROP initialization:'
        write (*,*) error_message
        stop
    end if
    wmm = wmol(comp_array)
    initialized = .true.
end subroutine initialize
```

```
subroutine CO2_TD(T, D, error_code, temp, pres, dens, enth, entr, ssnd)
    real(dp), intent(in) :: T, D
    integer, intent(out) :: error_code
    real(dp), intent(out), optional :: temp, pres, dens, enth, entr, ssnd
    real(dp) :: pres_RP, dens_mol, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap
    real(dp) :: qual, inte_mol, enth_mol, entr_mol, cv_mol, cp_mol, ssnd_RP
    if (.not. initialized) call initialize()
    dens_mol = D / wmm  ! convert density to molar basis
    call TDFLSH(T, dens_mol, comp_array, pres_RP, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap, &
                qual, inte_mol, enth_mol, entr_mol, cv_mol, cp_mol, ssnd_RP, error_code, error_message)
    if (present(temp)) temp = T
    if (present(pres)) pres = pres_RP
    if (present(dens)) dens = D
    if (present(enth)) enth = enth_mol / wmm
    if (present(entr)) entr = entr_mol / wmm
    if (present(ssnd)) ssnd = ssnd_RP
end subroutine CO2_TD

subroutine CO2_TP(T, P, error_code, temp, pres, dens, enth, entr, ssnd)
    real(dp), intent(in) :: T, P
    integer, intent(out) :: error_code
    real(dp), intent(out), optional :: temp, pres, dens, enth, entr, ssnd
    real(dp) :: dens_mol, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap
    real(dp) :: qual, inte_mol, enth_mol, entr_mol, cv_mol, cp_mol, ssnd_RP
    if (.not. initialized) call initialize()
    call TPFLSH(T, P, comp_array, dens_mol, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap, &
                qual, inte_mol, enth_mol, entr_mol, cv_mol, cp_mol, ssnd_RP, error_code, error_message)
    if (present(temp)) temp = T
    if (present(pres)) pres = P
    if (present(dens)) dens = dens_mol * wmm
    if (present(enth)) enth = enth_mol / wmm
    if (present(entr)) entr = entr_mol / wmm
    if (present(ssnd)) ssnd = ssnd_RP
end subroutine CO2_TP

subroutine CO2_PH(P, H, error_code, temp, pres, dens, enth, entr, ssnd)
    real(dp), intent(in) :: P, H
    integer, intent(out) :: error_code
    real(dp), intent(out), optional :: temp, pres, dens, enth, entr, ssnd
    real(dp) :: temp_RP, dens_mol, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap
    real(dp) :: qual, inte_mol, enth_mol, entr_mol, cv_mol, cp_mol, ssnd_RP
    if (.not. initialized) call initialize()
    enth_mol = H * wmm  ! convert enthalpy to molar basis
    call PHFLSH(P, enth_mol, comp_array, temp_RP, dens_mol, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap, &
                qual, inte_mol, entr_mol, cv_mol, cp_mol, ssnd_RP, error_code, error_message)
    if (present(temp)) temp = temp_RP
    if (present(pres)) pres = P
    if (present(dens)) dens = dens_mol * wmm
    if (present(enth)) enth = H
    if (present(entr)) entr = entr_mol / wmm
    if (present(ssnd)) ssnd = ssnd_RP
end subroutine CO2_PH

subroutine CO2_PS(P, S, error_code, temp, pres, dens, enth, entr, ssnd)
    real(dp), intent(in) :: P, S
    integer, intent(out) :: error_code
    real(dp), intent(out), optional :: temp, pres, dens, enth, entr, ssnd
    real(dp) :: temp_RP, dens_mol, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap
    real(dp) :: qual, inte_mol, enth_mol, entr_mol, cv_mol, cp_mol, ssnd_RP
    if (.not. initialized) call initialize()
    entr_mol = S * wmm  ! convert entropy to molar basis
    call PSFLSH(P, entr_mol, comp_array, temp_RP, dens_mol, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap, &
                qual, inte_mol, enth_mol, cv_mol, cp_mol, ssnd_RP, error_code, error_message)
    if (present(temp)) temp = temp_RP
    if (present(pres)) pres = P
    if (present(dens)) dens = dens_mol * wmm
    if (present(enth)) enth = enth_mol / wmm
    if (present(entr)) entr = S
    if (present(ssnd)) ssnd = ssnd_RP
end subroutine CO2_PS

subroutine CO2_HS(H, S, error_code, temp, pres, dens, enth, entr, ssnd)
    real(dp), intent(in) :: H, S
    integer, intent(out) :: error_code
    real(dp), intent(out), optional :: temp, pres, dens, enth, entr, ssnd
    real(dp) :: temp_RP, pres_RP, dens_mol, dens_liq_mol, dens_vap_mol, comp_array_liq, comp_array_vap
    real(dp) :: qual, inte_mol, enth_mol, entr_mol, cv_mol, cp_mol, ssnd_RP
    if (.not. initialized) call initialize()
    enth_mol = H * wmm  ! convert enthalpy to molar basis
    entr_mol = S * wmm  ! convert entropy to molar basis
    call HSFLSH(enth_mol, entr_mol, comp_array, temp_RP, pres_RP, dens_mol, dens_liq_mol, dens_vap_mol, &
                comp_array_liq, comp_array_vap, qual, inte_mol, cv_mol, cp_mol, ssnd_RP, error_code, error_message)
    if (present(temp)) temp = temp_RP
    if (present(pres)) pres = pres_RP
    if (present(dens)) dens = dens_mol * wmm
    if (present(enth)) enth = H
    if (present(entr)) entr = S
    if (present(ssnd)) ssnd = ssnd_RP
end subroutine CO2_HS


end module CO2_properties
```

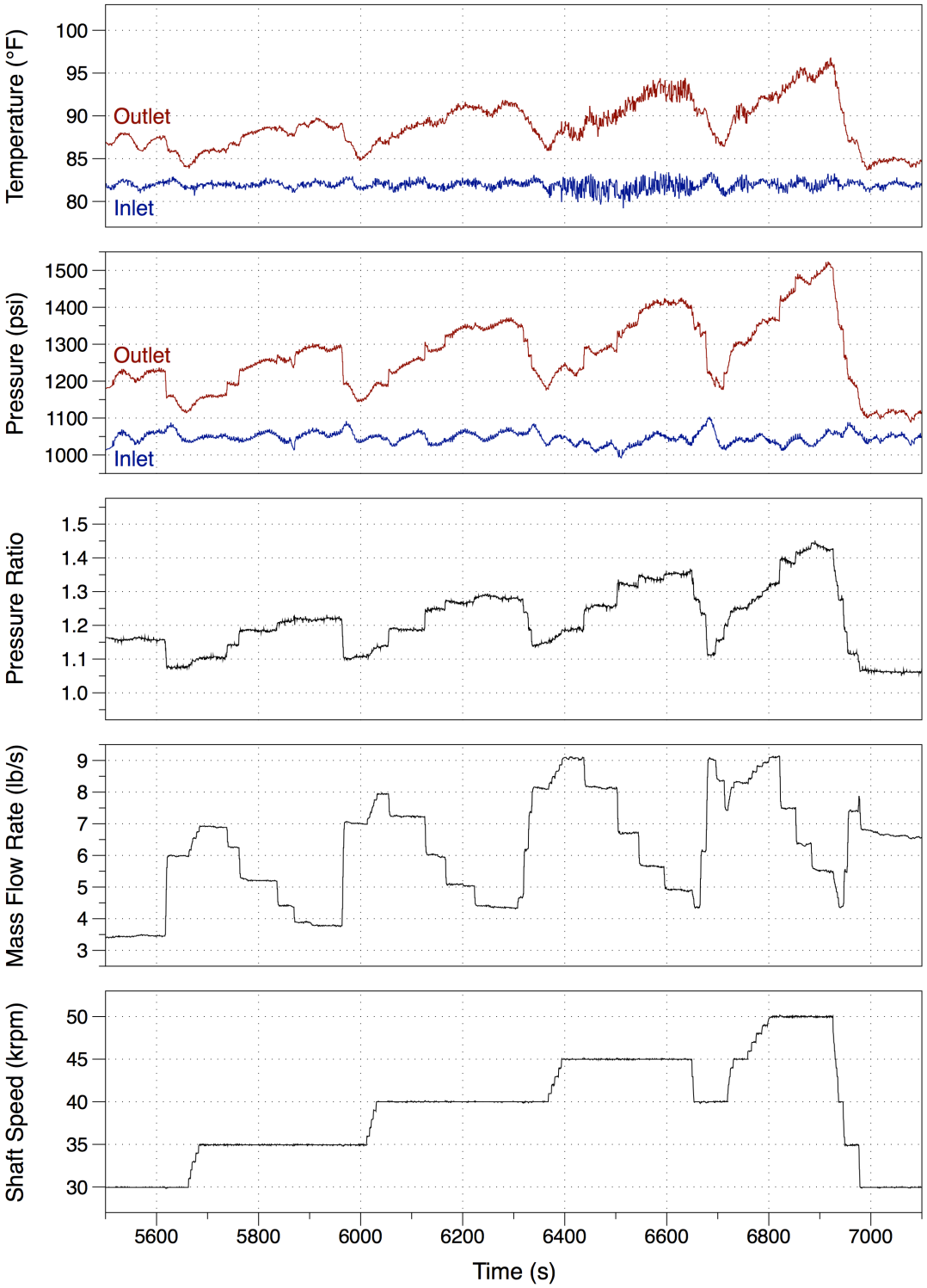
Appendix VIII: Recorded Data Plots for SNL Compression Test Cases



Supercritical Test Case



Liquid Test Case



Gas Test Case