1. Overview

The *Python* code described in this document interfaces with NREL's supercritical carbon dioxide (sCO_2) power cycle design [1], cost [2], and steady state off-design performance models [3]. The sCO_2 model provides heat to the cycle through a counter-flow primary heat exchanger employing a sensible heat heat-transfer fluid (HTF), rejects heat with an air-cooler, and contains simple, recompression, and partial cooling cycle configurations. The recompression cycle off-design model is more thoroughly tested than the simple and partial cooling configurations. There are two additional design-point cycle configurations which do not include off-design modeling: recompression with HTR bypass and turbine split flow. The sCO_2 performance code is part of NREL's SAM Simulation Core (SSC) that contains the technology models for SAM. This document describes i) high-level modeling assumptions, model parameters, and functions and ii) provides examples for common cycle design (Section 2) and off-design (Section 3) analyses.

1.1. System Requirements

This *Python* code requires the following *Python* packages: *numpy, matplotlib,* and *pandas*. You can use *Conda* to run *sco2_sam_python_environment.yml* to create a *Python* environment with compatible versions of *Python* and these packages. You may also get the version numbers from the *yml* file and install the packages using a different tool. Presently, this code only works on Windows, because of how it interfaces with the compiled SSC code.

1.2. Getting Started

Download or clone NREL's System Advisor Model (SAM) [4] repository from GitHub (https://github.com/NREL/SAM). The code described in this document is in the subfolder *samples/CSP/sco2_analysis_python_V2*.

The *core* folder contains *Python* methods that support configurating a simulation, handling output data, and plotting. This folder also must contain a copy of *ssc.dll* to access the performance code in SSC.

The *examples* folder contains *examples_main.py*, described in Section 2, and *off_design_examples.py*, described in Section 3. You should be able to run both files without modifying them, and they should create output files and plots in the *examples* folder.

2. Design Point Modeling

The design point model uses a fixed design-point isentropic efficiency to define turbomachinery performance, approach temperatures to specify the compressor inlet temperature (versus the design-point ambient temperature) and turbine inlet temperature (versus the design-point HTF hot temperature), and either minimum temperature difference, effectiveness or conductance to specify characterize counterflow recuperator performance. The following subsections explain different options to define and optimize remaining cycle parameters like pressures, split fraction, and recuperator conductance allocation. Neises and Turchi [1] present a cycle design optimization study that provides an example of how this model can be used for technoeconomic analyses.

2.1. Design Parameters

The function **get_sco2_design_parameters()** returns a Python *dictionary* containing default values for all cycle model parameters[1]. Many of the parameters define component design-point performance and are well described by the comments in the function. The following parameters require a more detailed explanation.

Cycle Configuration

The design parameter **cycle_config** defines whether the cycle is a recompression cycle (1), partial cooling cycle (2), recompression with HTR bypass (3), or turbine split flow (4). The code can model the simple cycle when **cycle_config** represents the recompression cycle and the design parameter **is_recomp_ok** is set to 0 (see below).

Recuperator design performance

The design model uses the design parameter **design_method** to provide several different options to define the recuperator performance. The default value of **design_method** is **2**, and in this case the model uses the total recuperator conductance set in design parameter **UA_recup_tot_des** and optimizes how much is allocated to each the low temperature and high temperature recuperator. If **design_method** is **3**, then the model uses heat exchanger design parameters to set the performance of each recuperator. For example, the design parameter **LTR_design_code** lets the user specify either the LTR conductance (1), minimum temperature difference (2) or the design effectiveness (3). If **design_method** is **1**, then the model finds the total recuperator conductance required to achieve the cycle efficiency set in design parameter **eta_thermal_des**. The optimizer will determine how much of the total conductance to allocate to each the low temperature and high temperature recuperator. Note the turbine split flow configuration does not support design method 1.

Free parameters

- **is_recomp_ok** defines whether the recompressor fraction is optimized (=1), set to zero to model a simple cycle (=0), or fixed (<0, where the model sets the recompression fraction to the absolute value of **is_recomp_ok**).
- **is_P_high_fixed** defines whether the main compressor outlet pressure is optimized (=0) or fixed (=1, where the model sets the main compressor outlet pressure to the design parameter **P_high_limit**). We have found that for typical values for the upper pressure limit (~20-35 MPa), the model finds the optimal design pressure at the upper pressure limit. In cases where the optimizer chooses a lower pressure as the optimal value, usually the efficiency improvement is negligible while the power density significantly decreases.
- **is_PR_fixed** defines whether the low pressure compressor (main compressor in the recompression cycle and precompressor in the partial cooling cycle) inlet pressure is optimized (=0), set to a fixed pressure ratio versus the main compressor outlet pressure (>0 where the pressure ratio is the input value), or fixed (<0, where the model sets the inlet pressure to the absolute value of **is_PR_fixed**).
- **is_IP_fixed** defines for the partial cooling cycle whether the pre-compressor outlet is optimized (=0), set to a fixed pressure ratio versus the main compressor outlet pressure (>0 where the pressure

[1] These parameters do not define any specific commercial or proposed system but rather a set of reasonable target values based on the literature. Our intent is to have a default model and simulation results to which we can compare when modifying the underlying performance code.

ratio is the input value), or fixed (<0, where the model sets the inlet pressure to the absolute value of **is_IP_fixed**).

- **is_bypass_ok** defines whether the bypass fraction is optimized (=1), set to zero to remove the bypass heat exchanger, or fixed (<0, where the model sets the bypass fraction to the absolute value of **is_bypass_ok**).
- **is_turbine_split_ok** defines the split fraction for the turbine split flow cycle, where (=1) is optimized, and (<0) is fixed to the absolute value of **is_turbine_split_ok**.

2.2. Examples in *design_point_examples.py*

Cycle design simulation with default parameters

The simplest analysis is to solve the cycle design for pre-defined "default" design parameters. First, create an instance of the **C_sco2_sim** class. Next, pass a copy of the design parameters from **get_sco2_design_parameters** using the class member **overwrite_default_design_parameters**. Then, run the design simulation through the class member **solve_sco2_case()**. Simulation results are stored as a *dictionary* in the class member **m_solve_dict**, and member **m_solve_success** reports whether the performance code solved without throwing an exception. View these values by printing them. Finally, use the method **save_m_solve_dict** to save results to a file. **C_sco2_sim** initializes with instructions to only save the results in *json* format, but you can also save in *csv* format by changing member variable **m_also_save_csv** to *True*.

Plotting a cycle design

First, create an instance of the **C_sco2_TS_PH_plot(result_dict)** class and use a saved design solution *dictionary* as the argument. To automatically save the plot when you create it, set the class member **is_save_plot** to *True* and assign a file name to class member **file_name**. Use class method **plot_new_figure()** to create the plot. You can use classes **C_sco2_cycle_TS_plot** and **C_sco2_cycle_PH_plot** to create individual TS and PH plots, respectively. If you change design parameters significantly from the default values, you may have to adjust the axis limits or annotation formatting.

Modifying the cycle design parameters to fully constrain the cycle design

After you create an instance of the **C_sco2_sim** class, you can overwrite its baseline design parameters with a *dictionary* of parameters you want to change. Design parameters that are not included in your *dictionary* keep their previously defined default values. For example, you may want to fix the pressures, recompression fraction, and recuperator parameters that are typically set by the optimizer to values from previous simulations or another reference. Use a partial dictionary to fix the recompression fraction, fix the pressure ratio, change the design method, and design recuperators to hit a target effectiveness. Next, use the **C_sco2_sim** method **overwrite_des_par_base(dictionary_new_parameters)** to modify the baseline design parameters. Then, you can run the simulation and view, save, and plot your results. Note that rerunning **solve_sco2_case()** will overwrite calculated class member like **m_solve_dict**.

Comparing two cycle designs

You can compare two cycle designs by processing the data in the saved *json* or *csv* files. You also can create a plot that overlays both cycles on TS and PH diagrams. Create an instance of the **C_sco2_TS_PH_overlay_plot(result_dict1, result_dict2)** class and use the saved design solution *dictionaries* as the arguments. Set the class member **is_save_plot** to *True* to save the plot. This class names the plot file based on the values of important cycle design parameters. Use class method **plot_new_figure()** to create the plot. You may need to adjust the axis limits or annotation formatting.

Running a parametric study

The **C_sco2_sim** class method **solve_sco2_parametric(list_of_partial_dictionaries)** runs a parametric using the modified cases in the input argument that is a *list* where each element is a *dictionary* that modifies some default design parameters. *Dictionary* elements in the *list* do not need to change the same parameters or even the same number of parameters, although organizing the parametrics this way may make the results easier to understand. The parametric simulation results are stored in the class member **m_par_solve_dict**. Each item in **m_par_solve_dict** *dictionary* is a *list* with a length equal to the number of parametric runs. Each element in the *list* corresponds to the item variable type, so variables that are themselves *lists* (e.g. ‘P_state_points’) are *lists* of *lists*. Use the method **save_m_par_solve_dict** to save results to a file. To save in *csv* format change member variable **m_also_save_csv** to *True*. Each column in the *csv* represents a single simulation in the parametric study.

Plotting a 1D parametric study

You can plot 1D parametric studies using the class **C_des_stacked_outputs_plot(list_of_parametric_dictionaries)**. The input argument requires the parametric solution *dictionary* is in *list* format, because this class can overlay multiple parameteric solutions. This class uses variable label and unit conventions defined in **get_des_od_labels_unit_info__calc_metrics()** in *sco2_cycle_ssc.py*. Note that the class member **x_var** uses “recup_tot_UA” to define the total recuperator conductance, which is different than the string used to set the corresponding design parameter (“UA_recup_tot_des”). The class member **y_var** sets the dependent variables, and each will have its own subplot. The class member **max_rows** determines how the subplots are configured on the figure. There are several other class members in *sco2_plot.py* that allow you to adjust formatting. Set **is_save** to *True* and specify **file_name** to save the plot, and use **create_plot()** to generate the plot.

3. Off-Design Performance Modeling

The off-design model explores how a fixed cycle design responds to changes to external inputs (HTF inlet temperature, HTF mass flow, and ambient temperature). The model assumes that the low-pressure compressor inlet pressure is adjustable. (We implicitly assume this pressure is adjusted via inventory control but do not model the process.) The model also assumes that air-cooler fan speed is adjustable to control the main compressor inlet temperature in the recompression cycle and both the main compressor and precompressor inlet temperatures in the partial-cooling cycle. The cycle off-design model has routines to maximize net cycle power output while constraining the high-side pressure and air-cooler fan power to their respective design values and fixing the cycle HTF outlet temperature to its design value. Alternatively, the user can choose to set these values and solve for cycle performance. The off-design model can also optionally optimize compressor shaft speeds or let the user set the off-design shaft speed. Neises 2020 [3] describes the recompression cycle off-design methodology in detail, discuses off-design performance versus ambient temperature and HTF mass flow rate, and explores the influence of shaft speed control. The partial cooling off-design model uses the same component models and follows the same off-design approach, but it presently is not documented or thoroughly tested. Note the recompression with HTR bypass and turbine split flow configurations do not support off-design modeling.

3.1. Code Organization

An off-design simulation necessarily requires a design point, so the code enables the user for a given cycle design to simulate multiple off-design scenarios in a single call to the sCO_2 performance code via the **solve_sco2_case()** method. Simulation outputs are stored in **C_sco2_sim** class member **m_solve_dict**. Off-design outputs that are a single value for each off-design simulation (e.g. net thermal efficiency) are reported as an array with a length equal to the number of off-design simulations. Similarly, off-design outputs that are an array value for each off-design simulation are reported as a matrix with the number of rows equal to the number of off-design simulations. The following subsections describe different ways to call the off-design model and refer to examples in *off_design_examples.py*.

3.2. Simulation Speed Dependence on Convergence Tolerance Influence

The cycle off-design solution shown in Figure 14 in Neises 2020 [3] uses a relative tolerance to converge. The results in the paper were generated using a default value of 0.001. However, off-design simulations using this value for convergence tolerance can be prohibitively slow, especially when the model is optimizing shaft speeds. The input parameter **od_rel_tol** sets the tolerance as $10^{-\boldsymbol{od\_rel\_tol}}$. That is, an input value of 3 corresponds to the default tolerance of 0.001. Decreasing this value to around 2.5 enables faster simulations at the expense of numerical accuracy. We recommend plotting simulation results (in all cases, but especially when using a tolerance greater than 0.001) to check for aberrations. We also recommend checking solutions of interest by re-running them with the default tolerance (using the optimized control values from the faster, higher tolerance solution as inputs in **od_set_control** as described in Subsection 3.4).

3.3. Off-Design Solution w/ Optimized Control

The **od_cases** input is a matrix composed of arrays of off-design inputs. For each array, the model maximizes net cycle power output while constraining the high-side pressure and air-cooler fan power to their respective design values and fixing the cycle HTF outlet temperature to its design value. Each array also includes an input to specify whether the model should optimize individual shaft speeds (note that at the default convergence tolerance, a single off-design solution optimizing shaft speeds can take over an hour). An array must be defined by index as follows:

0) HTF inlet temperature in degrees Celsius
1) Normalized HTF mass flow (i.e. the design HTF mass flow rate = 1.0)
2) Ambient air temperature in degrees Celsius
3) Recompressor shaft speed
    - =1: fix at design
    - =0: optimize
    - <0: fix to normalized (against design) value equal to the absolute value of the input
4) Main compressor shaft speed
    - =1: fix at design
    - =0: optimize
    - <0: fix to normalized (against design) value equal to the absolute value of the input
5) Pre-compressor shaft speed
    - =1: fix at design
    - =0: optimize

- <0: fix to normalized (against design) value equal to the absolute value of the input

Example 1 in *off_design_examples.py* demonstrates how to call this off-design solution method and process the results.

3.4. Off-Design Solution w/ User-Specified Control Parameters

The **od_set_control** input is a matrix composed of arrays of off-design inputs. For each array, the model finds the off-design solution by applying the controls from the input array. An array must be defined by index as follows:

0) HTF inlet temperature in degrees Celsius
1) Normalized HTF mass flow (i.e. the design HTF mass flow rate = 1.0)
2) Ambient air temperature in degrees Celsius
3) Low-pressure compressor inlet pressure in MPa
4) Main-compressor inlet temperature in degrees Celsius
5) Pre-compressor inlet temperature in degrees Celsius
6) Recompressor shaft speed
    - =1: fix at design
    - <0: fix to normalized (against design) value equal to the absolute value of the input
7) Main compressor shaft speed
    - =1: fix at design
    - <0: fix to normalized (against design) value equal to the absolute value of the input
8) Pre-compressor shaft speed
    - =1: fix at design
    - <0: fix to normalized (against design) value equal to the absolute value of the input

Because this solution method is relying on inputs for the low-pressure compressor inlet pressure and temperature, it is possible that the inputs will result in at least one of the following outcomes:

- A compressor operates in surge. This occurs when the output *mc_phi_od* is less than *mc_phi_surge* or the output *rc_phi_od* is less than *rc_phi_surge*. In surge conditions, the model uses the fabricated compressor data described in Section 3 of Neises 2020 [3] to converge the cycle solution, but the solution is considered infeasible.
- The main compressor outlet pressure (*P_mc_out_od*) exceeds its design value (*P_comp_out*).
- Air-cooler fan power (*mc_cooler_W_dot_fan_od, pc_cooler_W_dot _fan_od*) exceeds its design value (*mc_cooler_W_dot_fan, pc_cooler_W_dot_fan)*.
- The HTF cold temperature (*T_htf_cold_od*) differs from its design value (*T_htf_cold_des*). This outcome does not necessarily imply infeasibility, but it likely corresponds to a different control strategy than used in **od_set_control** and presented in Neises 2020 [3].

Example 2 in *off_design_examples.py* demonstrates how to call this off-design solution method and process the results.

4. References

[1] T. Neises and C. Turchi, "Supercritical carbon dioxide power cycle design and configuration optimization to minimize levelized cost of energy of molten salt power towers operating at 650°C," *Sol. Energy*, vol. 181, pp. 27–36, 2019.

[2] M. D. Carlson, B. M. Middleton, and C. K. Ho, “Techno-Economic Comparison of Solar-Driven sCO2 Brayton Cycles Using Component Cost Models Baselined with Vendor Data and Estimates,” in *Proceedings of the ASME 2017 11th International Conference on Energy Sustainability*, 2017, pp. 1–7.

[3] T. Neises, “Steady-State Off-Design Modeling of the Supercritical Carbon Dioxide Recompression Cycle for Concentrating Solar Power Applications with Two-Tank Sensible-Heat Storage,” *Sol. Energy*, 2020.

[4] National Renewable Energy Laboratory, “System Advisor Model Version 2020.02.29,” 2020. [Online]. Available: https://sam.nrel.gov/download.