**System Requirements**

These Python files were written for Python 3.7.0 and will not work with Python 2 installations. The files also require several add-on packages that I downloaded with the Anaconda3 installation, but there are other options to acquire these packages. Presently, this code only works on Windows, because of how it interfaces with NREL's compiled C++ code.

**Getting Started**

Download the GitHub code and save it locally.

**Summary**

This code interfaces with NREL’s supercritical carbon dioxide (sCO2) power cycle design, cost, and off-design performance models. The sCO2 models are part of NREL’s SAM Simulation Core (SSC) that contains the technology models for the System Advisor Model (SAM). This code allows the user to explore the sCO2 in more detail than possible when using the sCO­­2 cycle option in the Molten Salt Power Tower model. In particular, this code:

* Allows the user to set more design variables than the SAM User Interface.
* Easily run, save, and plot parametric analyses on design variables.
* Use component cost correlations from Carlson to calculate component and total cycle cost.

The code allows the user to select between the simple, recompression, and partial cooling cycle configurations. It uses simple isentropic efficiency equations for turbomachinery performance. The recuperator model assumes a counterflow configuration, and it discretizes each heat exchanger to solve its performance for a given conductance. For fixed design parameters, including total recuperator conductance and high-side pressure, the design model optimizes the low-pressure compressor inlet pressure, recompression fraction (recompression and partial cooling cycles), and intermediate pressure (partial cooling cycle) to maximize cycle efficiency.

This document provides examples for common cycle analyses and is a working document. Please send questions and bugs to [ty.neises@nrel.gov](mailto:ty.neises@nrel.gov).

**Code Organization**

The *core* folder contains Python methods that streamline configurating a simulation, handling output data, and plotting. This folder also contains the compiled SSC code that performs the sCO2 cycle simulation.

The *examples* folder contains “*examples\_main.py”*, a Python file that performs common sCO2 analyses by calling methods contained in the files in the *core* folder. The examples describe the code in this file.

**Examples**

Cycle design simulation with default parameters

The simplest analysis is to solve the cycle design for pre-defined “default” design parameters. (These parameters do not define any specific commercial or proposed system but rather a set of reasonable values based the literature. Our intent is to have a default model and simulation results to which we can compare when modifying the underlying performance code.) First, create an instance of the **C\_sco2\_sim** class. When the class initializes, it sets the default sCO2 model design parameters to the values in the function **get\_default\_sco2\_dict()**. 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 successfully. 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 save 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

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 default values as defined in the function **get\_default\_sco2\_dict()**. 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 calculate class member like **m\_solve\_dict**.

Comparing two cycle designs

You can compare two cycle designs by either processing the data in the saved json or csv files. You also can create a plot with 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 save 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 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** 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 list format, because this class can overlay multiple parameteric solutions (more on this later). 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 call member **x\_var** uses “T\_HTF” to define the HTF temperature, which is different than the string used for the design parameter (“T\_htf\_hot\_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.