The International Supercritical CO_2 Energy Technologies Symposium
March 2 - 5, 2026, Pittsburgh, PA
Poster 048

Modeling sCO_2 Power Cycles with Twine: A Composable, Open-Source Framework in Rust

John Dyreby and Greg Troszak
Isentropic Development
Verona, WI

ABSTRACT

Twine is an open-source framework for functional and composable system modeling. The framework is written in Rust and uses deterministic component models with strong typing and explicit interfaces. These features make models easier to test and reduce problems that occur when models become large or are modified over time. To show how the approach applies to thermodynamic cycle analysis, a supercritical carbon dioxide (sCO_2) Brayton cycle with recompression was implemented in Twine. The model is based on an earlier Fortran version developed by the author and can predict design-point and off-design performance. Results from Twine agree well with the earlier model and the new implementation provides a more modern foundation for future work through a flexible, problem-based solution approach. The full Twine framework and the recompression cycle models are available as open-source code for use in sCO_2 cycle studies and related energy system simulations.

INTRODUCTION

Supercritical carbon dioxide (sCO_2) Brayton cycles continue to be investigated for a range of applications, including concentrating solar power, next-generation nuclear systems, and industrial waste-heat recovery. Predicting design-point and off-design performance is an important part of evaluating cycle options and understanding how they behave under different operating conditions. The author previously developed a set of recompression cycle models for these purposes [1, 2], implemented in Fortran and used in several design and control studies.

Although those models remain functional, they share drawbacks common to Fortran code written by an engineering doctoral student. A number of assumptions were made that require careful handling of data flow and internal state, unit handling must be managed manually, and modifying or extending the models requires careful attention to avoid introducing inconsistencies. As models grow in size and scope, or are adapted for other studies, these issues become more noticeable. In addition, building and running the current Fortran versions requires an older Python toolchain and f2py workflow, which makes the models more difficult to maintain and use in modern environments.

Twine was created in part to address some of these challenges. Twine is a general-purpose system modeling framework that uses a functional approach, where components are represented as pure functions with strongly typed inputs and outputs. The intent is to reduce hidden dependencies, make interfaces clearer, and improve the overall reliability of models used for engineering analysis. While Twine is not specific to sCO_2 systems, the recompression cycle is a useful demonstration because it

includes several interacting components, flow splits, and sensitivity to thermophysical properties.

The purpose of this paper is twofold. First, the main features of Twine that are relevant to thermodynamic system modeling are summarized. Second, a recompression cycle model based on the earlier Fortran version is presented. The comparison shows that Twine can reproduce established results while providing a modern and more maintainable platform for future work. In addition, Twine couples system models with a general solution framework that supports equation solving, optimization, and transient analysis without requiring changes to component implementations.

TWINE FRAMEWORK OVERVIEW

Twine represents system models as compositions of pure functions. Each component model takes a defined set of inputs and returns outputs without using shared state or side effects. This type of structure makes it easier to understand how information moves through a model and simplifies the process of testing individual components. It also avoids a number of problems that arise when components update internal state or rely on global variables.

A key part of Twine's approach is strong typing. Inputs and outputs are represented using explicit types for quantities such as temperatures, pressures, mass flow rates, and thermodynamic states. Units are enforced at compile time, which prevents dimensional mismatches and makes it clearer how components connect to one another. Unit conversions are handled automatically through the type system, removing the manual bookkeeping that can lead to incorrect numerical values. When a model is modified, the compiler can catch inconsistencies between components before they become runtime issues.

Twine distinguishes between the definition of a system model and the numerical problems posed on top of that model. A *Model* specifies how components are connected and how information flows through the system, while separate *Problem* definitions describe how the model is evaluated or solved. This separation allows the same system model to be reused for different types of analyses without modifying component implementations.

Numerical solution capabilities are provided by the `twine-solve` library, which defines trait-based `Problem` interfaces for root-finding, optimization, and transient analysis. Equation problems are used to solve for unknown quantities that satisfy system-level constraints, such as matching heat exchanger conductance or determining mass flow rate at an operating point. Optimization problems are used to identify operating conditions that maximize or minimize a specified objective, such as thermal efficiency. Transient problems extend this approach to time-dependent analyses, allowing a model to be evaluated over a sequence of operating conditions. This structure keeps component models simple while supporting a range of system-level analyses.

Although Twine is in early development, it includes a growing set of components useful for system-level modeling. These include thermal elements such as heat exchangers and storage tanks, turbomachinery models for compressors and turbines, controllers for basic control tasks, and schedule components for time-based operations. Twine also defines an abstraction layer for thermodynamic property evaluation so that property models such as REFPROP, CoolProp, or table-based libraries can be used without modifying the rest of the system model.

The framework is not limited to thermodynamic cycles. The same modeling approach can be used for a broad range of applications. The recompression cycle is used here simply because it is a representative example of a system that benefits from clear interfaces and reliable property evaluation.

RECOMPRESSION CYCLE MODELING

The recompression sCO_2 Brayton cycle includes the main compressor, recompression compressor (recompressor), turbine, high- and low-temperature recuperators, and mixing/diverting valves. A fraction of the flow bypasses the low-temperature recuperator to improve overall recuperation effectiveness. The cycle is sensitive to accurate property evaluation, especially near the critical region, and involves several interacting components, making it a suitable test for Twine.

The Twine implementation follows the structure of the author's earlier Fortran models [1, 2]. Each major component is represented by a function with clearly defined inputs and outputs. Flow splits and merges are represented explicitly so that mass and energy balances are directly enforced. Turbomachinery performance at the design point is represented using constant isentropic efficiency. Off-design performance is based on characteristic curves similar to those used in the previous work.

Twine's fluid property model abstraction allows the same cycle model to run with different CO_2 property libraries. This capability provides flexibility for evaluating property models with different accuracy or performance characteristics.

The capabilities provided by `twine-solve` are used extensively in the recompression cycle model. For example, the recuperator, modeled as a series of sub-heat exchangers, is formulated as an equation problem and includes an observer that monitors node temperatures, allowing second-law violations to be detected during solution. Equation problems are also used to iterate mass flow rate to determine consistent off-design operating points. Optimization problems are applied to identify operating conditions that maximize thermal efficiency, and the same model structure is intended to support transient problems for evaluating optimally operated off-design performance over extended periods, such as a simulated year.

Overall, the Twine-based cycle model reproduces the structure and assumptions of the earlier implementation but benefits from clearer interfaces and improved maintainability. The functional structure also simplifies testing of individual components before assembling the full cycle.

RESULTS AND DISCUSSION

Design-point results from the Twine implementation agree well with the earlier Fortran model when equivalent assumptions and property libraries are used. Key performance quantities such as efficiency, compressor and turbine work, and heat exchanger duties show similar values. Minor differences are mainly related to variations in thermodynamic fluid property model implementation or numerical tolerances.

Off-design predictions also follow the same general trends as the earlier model. Changes in ambient temperature, compressor speed, or heat exchanger performance produce behavior consistent with the previous simulations. Because Twine separates component functions and system assembly, it is straightforward to modify or update off-design correlations, which should help future evaluations that rely on turbomachinery maps or other performance data.

While agreement with the earlier Fortran results provides confidence in the implementation, the primary contribution of this work is the modeling approach enabled by Twine rather than the specific numerical results for a single cycle configuration. The Twine framework provides a reusable foundation for constructing, solving, and extending thermodynamic system models using a consistent modeling approach. The recompression cycle model demonstrates how component libraries, property model abstraction, and flexible solve strategies can be combined to support design-point, off-design, and

optimization studies within a single framework.

By emphasizing composability, strong typing, and explicit problem formulation, Twine lowers the barrier to developing and maintaining complex cycle models. The resulting models are easier to test, modify, and extend, making the framework well suited for continued sCO_2 cycle development as well as for other thermodynamic and energy-system modeling applications.

CONCLUSIONS

Twine provides a functional and strongly typed framework for system modeling that helps improve clarity and reliability in thermodynamic cycle models. The recompression cycle example demonstrates that Twine can represent a realistic sCO_2 system and reproduce results from an earlier Fortran implementation. The modernized model and the Twine framework are available as open-source resources and are intended to support continued development of sCO_2 cycle analyses and other engineering simulations.

REFERENCES

1. **Dyreby, J., Klein, S., Nellis, G., and Reindl, D.** "Design Considerations for Supercritical Carbon Dioxide Brayton Cycles With Recompression." *Journal of Engineering for Gas Turbines and Power*, Vol. 136, No. 10, 2014, Paper No. GTP-14-1240, pp. 101701. https://doi.org/10.1115/1.4027936

2. **Dyreby, J. J.** *Modeling the Supercritical Carbon Dioxide Brayton Cycle with Recompression*. Ph.D. Dissertation, University of Wisconsin-Madison, 2014.