Received: 26 October 2020 | Revised: 22 April 2021 | Accepted: 24 April 2021

DOI: 10.1002/amp2.10095

RESEARCH ARTICLE

JOURNAL OF ADVANCED MANUFACTURING AND PROCESSING WILEY

The IDAES process modeling framework and model library—Flexibility for process simulation and optimization

Andrew Lee[1,2] | **Jaffer H. Ghouse**[1,2] | **John C. Eslick**[1,2] | **Carl D. Laird**[3] | **John D. Siirola**[3] | **Miguel A. Zamarripa**[1,2] | **Dan Gunter**[4] | **John H. Shinn**[4] | **Alexander W. Dowling**[5] | **Debangsu Bhattacharyya**[6] | **Lorenz T. Biegler**[7] | **Anthony P. Burgard**[1] | **David C. Miller**[1]

[1]National Energy Technology Laboratory, Pittsburgh, Pennsylvania, USA

[2]NETL Support Contractor, Pittsburgh, Pennsylvania, USA

[3]Sandia National Laboratories, Albuquerque, New Mexico, USA

[4]Lawrence Berkeley National Laboratory, Berkeley, California, USA

[5]University of Notre Dame, Notre Dame, Indiana, USA

[6]Department of Chemical and Biomedical Engineering, West Virginia University, Morgantown, West Virginia, USA

[7]Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Correspondence
Andrew Lee, National Energy Technology Laboratory, 626 Cochrans Mill Road, Pittsburgh, PA 15236, USA.
Email: andrew.lee@netl.doe.gov

Funding information
U.S. Department of Energy, Grant/Award Numbers: DE-FE0025912, DE-NA0003525, DEAC02-05CH11231

Abstract

Energy systems and manufacturing processes of the 21st century are becoming increasingly dynamic and interconnected, which require new capabilities to effectively model and optimize their design and operations. Such next generation computational tools must leverage state-of-the-art techniques in optimization and be able to rapidly incorporate new advances. To address these requirements, we have developed the Institute for the Design of Advanced Energy Systems (IDAES) Integrated Platform, which builds on the strengths of both process simulators (model libraries) and algebraic modeling languages (advanced solvers). This paper specifically presents the IDAES Core Modeling Framework (IDAES-CMF), along with a case study demonstrating the application of the framework to solve process optimization problems. Capabilities provided by this framework include a flexible, modifiable, open-source platform for optimization of process flowsheets utilizing state-of-the-art solvers and solution techniques, fully open and extensible libraries of dynamic unit operations models and thermophysical property models, and integrated support for superstructure-based conceptual design and optimization under uncertainty.

KEYWORDS

optimization, process modeling, Pyomo, simulation

1 | INTRODUCTION AND MOTIVATION

Over the next decade, hundreds of billions of dollars will be invested in next generation energy systems and industrial processes that will be more efficient, dynamically agile and interconnected. Shifts to more integrated production facilities will need to balance feedstock volatility, market pricing, and environmental impacts to support an increasingly circular economy. To address emerging opportunities and challenges, energy systems and industrial processes need to support greater innovation, including process intensification, hybrid systems

[The copyright line for this article was changed on 04 May 2021, after original online publication.]

concepts, and plant modularization. These new systems must have greater agility to dynamically respond to changing economic conditions, which may vary on an hourly or even sub-hourly basis. For example, electricity prices can vary widely due to mismatches in supply and demand, and industries that can shift production have a substantial incentive to coordinate with energy markets. Hybrid systems provide a broader opportunity to provide electricity during periods of high demand and shift production to other products when energy prices are low.

These needs further highlight the complex, dynamic design and operational challenges in the 21st century, which will require modeling and decision-making tools to maximize efficiency and operability over very wide ranges of operating conditions. Models will need to consider multiple temporal and spatial scales that balance short-term profitability with longer-term impacts of equipment wear and degradation, and explore interactions beyond the traditional boundaries of individual plants, including local, regional, and global markets.

1.1 | Modeling and optimization needs

To meet the requirements for designing the next generation of energy and industrial systems, process modeling and optimization tools for decision-making need to extend beyond traditional plant boundaries and consider interactions with the broader supply chain,[1] including the electricity grid. Large-scale optimization strategies are essential to design new processes incorporating multiple operating scenarios and dynamic response to changing conditions, which are defined through multi-scale linkages with models of the broader energy and supply chain systems.

Anticipating the development of new technologies and processes, the modeling framework must enable uncertainty quantification (UQ) in order to ensure robust designs and enable the reduction of technical risk through targeted experiments. Such capabilities will ultimately enable the technical performance guarantees essential for commercial deployment. Finally, since many of the characteristics of 21st century processes and energy systems will extend beyond current experience, new process synthesis strategies will be required to explore large design spaces that include novel technologies and phenomena as well as new configurations and operating paradigms.

2 | CURRENT PRACTICE AND ENABLING TOOLS

Current practice in computer-aided process engineering (CAPE) relies on a tool-chain that has been developed over the last 50 years. Early efforts in process simulation for CAPE began in the late 50s with the simple concept of connecting purpose-built unit models to form a flowsheet model. This led to the familiar sequential-modular (SM) simulation strategy where information flow of the unit calculations was sequenced based on the flowsheet structure, and common "utility models" (e.g., physical property calculations) could be called by the unit models. Early flowsheeting packages were first developed in-house at major chemical and petroleum companies followed by platforms by commercial vendors (e.g., DESIGN II,[2] HYSYS,[3] CHEMCAD,[4] PRO/II,[5] PROSIM[6]). The late 70s saw the development of the ASPEN simulator[7] which included more flexible data handling, improved phase separation routines and unit operations for solids handling. Nevertheless, the SM mode consists of procedural models where the model equations and solution strategy were strongly linked, leading to rigid calculation sequences. This can lead to time-consuming solution strategies with poor convergence behavior. These SM platforms have been steadily upgraded over the last four decades to accommodate advances in software engineering, such as objected-oriented architectures and graphical user interfaces, and they still remain the dominant mode for process simulation, design and analysis in engineering practice. However, formal optimization methods and advanced algorithms (i.e., for nonlinear programming (NLP) and mixed integer nonlinear programming (MINLP)) are rarely part of this workflow.

In parallel to the development of the modular mode, research in the 60s and 70s led to the equation-oriented (EO) simulation strategy, with academic packages such as SPEEDUP,[8] ASCEND[9] and QUASILIN.[10] Here the process model equations (describing the unit and utility models) are constructed independently of the solution procedure. Because of this, EO-based solution strategies allow for greater flexibility in design studies, and also in the creation and maintenance of specialized process models.

In contrast to modular simulators, EO methods are especially well suited for the construction of specialized process simulation models, especially with nonlinear (and possibly stiff) Differential Algebraic Equations (DAE).[11] These are needed because stiff DAE solvers (e.g., BDF) must often be applied, and these require Jacobian information from the DAE. The EO mode has been the main paradigm for more recent commercial simulators gPROMS[10,12,13] and Aspen Custom Modeler,[7] which have strong capabilities for modeling dynamic and distributed systems. Moreover, advanced DAE modeling platforms, such as DAEPACK[14] and ABACUSS[15] have been extended to deal with hybrid discrete/continuous dynamic simulation[16] and event location of state discontinuities. In addition, EO platforms

are well-suited for gradient-based optimization because of the ready availability of exact first and second derivatives and they can integrate directly with advanced mathematical programming tools.

On the other hand, EO methods require more expertise to solve large models, because a general-purpose Newton-based solver is typically used and complex initializations[17,18] must be supplied. For instance, Aspen Plus has been extended to deal with EO features in Aspen-EO where initialization is handled through partial solves within the SM mode. Nevertheless, due to the required expertise for EO modeling these approaches are not widely used for off-line modeling and design, where the modular approach provides an easier construction framework, especially for casual users. Instead EO modeling is widely used for specialized DAE and PDAE models, and purpose-built optimization models, for example, for real-time optimization.

On the other hand, the past two decades have seen significant progress in both computational power and large-scale numerical optimization algorithms, which greatly expand the types of problems which can be solved. Due to synergistic advances in computational algorithms, hardware and software, and automatic differentiation for EO models, advanced solution and optimization strategies can solve a wide range of challenging problems relevant to advanced process systems engineering including: model predictive control that involves dynamic nonlinear optimization,[19] superstructure-based conceptual design problems with integer variables,[20–22] and two-stage stochastic optimization problems involving uncertainty.[23] However, application of these strategies and tools has been largely limited to a few platforms (such as gPROMS[12] and Aspen Custom Modeler[7]), because many advanced algorithms developed by the optimization community are hard to integrate, and they perform less well with commercial simulation tools, particularly those in the SM mode.

To allow the realization of the full potential of state-of-the-art optimization algorithms, Algebraic Modeling Languages (AMLs) have been developed that can solve large-scale linear and nonlinear mathematical optimization problems with both continuous and discrete variables. Examples include AMPL,[24] GAMS,[25] AIMMS,[26] Pyomo,[27] and JuMP.[28,29] These modeling platforms require users to construct their own mathematical models for the systems they wish to study, and their open structure provides nearly limitless freedom in model construction and formulation, seamless calculation of exact first and second derivatives, and the integration of dozens of advanced optimization solvers. Nevertheless, for CAPE problems, AML platforms lack the specialized infrastructure and model libraries typically available in process simulators, and they often require models to be built from the ground up. This limitation becomes especially challenging for thermodynamic and physical property models. Additionally, most AMLs lack the modular and hierarchical structure generally expected by process engineers, which limits the reuse of models. In addition, they require advanced skills to build interconnected system models. Finally, the performance and reliability of the resulting model are heavily dependent on the engineer's expertise to avoid coding errors and properly formulate the optimization model. Taken together, these add a significant amount of time to the modeling process and require specialized knowledge that precludes widespread application.

As a result of these developments, there is currently no single platform which combines the infrastructure and modeling libraries required to address complex energy and chemical processes with integrated support for advanced optimization-based decision-making, that is, for (i) optimal steady-state and dynamic design and operations, including superstructure based conceptual design, and (ii) data reconciliation, parameter estimation, uncertainty quantification and optimization under uncertainty. This deficiency leads to greatly increased user effort and severely limits the effectiveness of process modeling and optimization workflows, as potential users must have access to and knowledge of multiple tools to handle each task.

2.1 | The IDAES integrated platform

The open-source Institute for the Design of Advanced Energy Systems (IDAES) Integrated Platform was developed to address these challenges and overcome the limitations of current process tools and enable next-generation process systems engineering applications.[30] This platform supports the full process modeling lifecycle from conceptual design to dynamic optimization and control within a single modeling environment. At the center of this platform is the Core Modeling Framework (IDAES-CMF) which leverages the open-source, extensible algebraic modeling environment, Pyomo.[27] Pyomo is written in the Python programming language, and it allows modelers to formulate, initialize, solve, and manipulate large-scale optimization problems with concise notation all within the Python ecosystem. Pyomo provides interfaces to dozens of optimization solvers, both open-source and commercial, and it supports executing solvers both locally and in distributed environments. Most importantly, Pyomo has the advanced modeling capabilities needed. Pyomo provides extensions for block-oriented modeling and Generalized Disjunctive Programming, explicit representation of network models, stochastic programming, and dynamic systems containing (partial) differential-algebraic equations. This

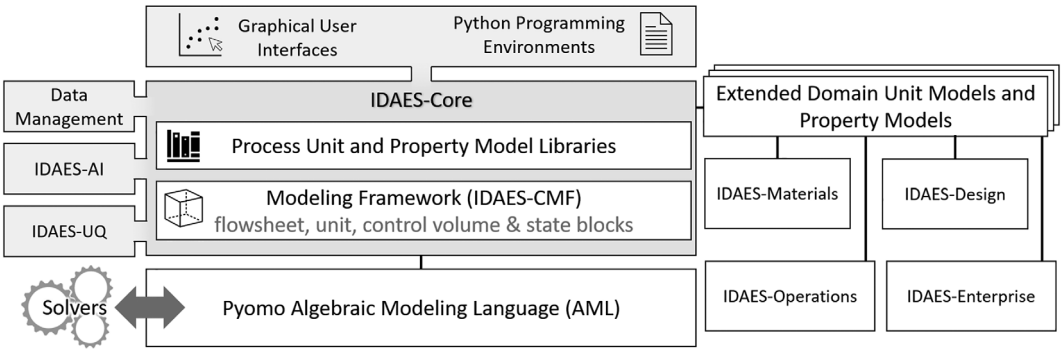

FIGURE 1 IDAES integrated platform

enables modeling flexibility and the consideration of constraints and objective functions in non-standard forms not easily handled by other modeling frameworks. New, custom algorithms and solution processes can be combined with these extensions to enable complex analyses. A diagram of the major elements of the IDAES Integrated Platform is shown in Figure 1.

This paper describes the IDAES-CMF and model libraries, demonstrating its capabilities for solving complex chemical engineering flowsheets through a case study. The IDAES-CMF reduces the overhead associated with building and customizing models by automating much of the workflow and providing several libraries of open, modifiable and flexible sub-models. Section 2 describes the IDAES-CMF along with its EO modeling structure, process model libraries and strategies for process flowsheet construction and initialization. Section 3 presents a case study on the modeling and optimization of the hydrodealkylation process, in order to describe the workflow and demonstrate the capabilities of the IDAES-CMF. Finally, Section 4 concludes the paper and summarizes the characteristics of the IDAES Integrated Platform and IDAES-CMF.

3 | THE IDAES CORE MODELING FRAMEWORK

As discussed above, two of the most significant limitations to the wide-spread adoption of EO approaches have been the significant amount of effort required in formulating models due to a lack of general-purpose model libraries, and the challenges initializing the overall model. With commercial process simulation tools including pre-built libraries of reliable models for most common unit operations, these tools have easily become the go-to solution for process modelers, with EO approaches limited to niche applications that process simulators cannot solve efficiently. The IDAES-CMF seeks to make EO approaches more accessible by addressing these concerns through providing a library of modular EO modeling elements which can be used to rapidly assemble complex process flowsheets.

The IDAES-CMF contains a library of models for common process unit operations and thermophysical properties which can be used to rapidly create process flowsheets. These models are internally consistent and interoperable and support different levels of complexity and accuracy for both steady-state and dynamic analysis, along with pre-built, generally applicable initialization routines. Additionally, the IDAES-CMF includes a library of customizable building blocks for constructing models of novel unit operations. These building blocks can be further combined with existing models to support multi-scale modeling. All of these building blocks also contain modular initialization routines that can be combined hierarchically to build reliable initialization schemes.

An important aspect of the entire IDAES-CMF is that every model and equation is completely open and extensible. Not only can a modeler see exactly what variables and constraints make up the model, but they also have complete freedom to modify these models by adding and removing variables and constraints, or even modifying those constraints which already exist within the model. This level of flexibility allows modelers to fully customize the model library to their needs, giving them full control of the rigor, complexity and tractability of the final model. This in turn allows the IDAES-CMF model libraries to be applied to a wide range of different problems, from conceptual design where complexity often needs to take second place to tractability, to dynamic control studies where complex non-linear performance curves must be considered.

3.1 | IDAES model libraries

The IDAES-Core model library includes common unit operations and thermophysical property models that

were formulated for optimization of both steady-state and dynamic processes. Table 1 shows the unit operation and property models currently available in the core model libraries.

These model libraries are constructed using the modular building blocks in the core modeling libraries (discussed below) in the same way user-defined models would be. These models are fully open and modifiable and are intended to provide a set of equations required to describe a generic version of a given unit operations. Modelers are fully encouraged to build upon these models by adding additional levels of detail such as efficiency and performance correlations to suit the needs of their application. Each model also incorporates a dedicated initialization routine.

TABLE 1 Unit models available in the core libraries

Unit model	
Distillation column and trays	Tray columns with phase equilibria on each tray. Including optional condenser and reboiler, arbitrary feeds and side draws.
Feed	Unit model for representing material feeds. Optional flash phase equilibrium of defined feed.
Flash drum	Flash distillation, supports multiple forms (PT, PH, etc.).
Heat exchanger	0-D and 1-D forms available, multiple options for calculation of temperature driving force.
Heater	Single 0-D material flow with heating or cooling, half of a heat exchanger.
Mixer	Arbitrary number of inlets, equal pressure and pressure minimization forms.
Pressure changer	Supports both compression and expansion. Includes options for isothermal, isentropic and liquid pumps with user-defined efficiency.
Product	Unit model for representing product streams.
Separator/splitter	Arbitrary number of outlets, allows for separation by component and/or phase.
Reactor unit models	
CSTR (well mixed vessel)	Extent of reaction equal to reaction rate multiplied by reactor volume.
Equilibrium reactor	Equilibrium assumption based on equilibrium coefficients.
Gibbs reactor	Equilibrium assumption based on minimizing Gibbs energy.
Plug flow reactor	1-D fluid flow with rate-based reactions.
Stoichiometric/ yield reactor	Reactor with user-defined extents of reaction.
Thermophysical property models	
Ideal gases and liquids	Ideal gas equation, ideal mixing rules for properties.
Cubic equations of state	Generic form with implementation of PR and SRK.
Activity coefficient methods	Supports NRTL and Wilson activity coefficient methods
Helmholtz equations of state	Pure component properties for water and CO_2

3.2 | IDAES building blocks

One of the greatest strengths of EO modeling tools is the ability for users to create custom models for novel applications; however, the effort required to first build a tractable model and then find a feasible initial solution has generally limited the application of custom models to academic and dedicated research applications. To address this limitation, the IDAES-CMF provides a suite of modular building blocks that utilize the hierarchical structure shown in Figure 2.

A process is considered to be a network of material "states," representing both the extensive and intensive properties of a material within the process at a given point in space and time. A unit model (i.e., model of a specific piece of equipment) consists of two or more material states that are linked together by sets of material, energy and momentum balances which describe how the material transitions from one state to another, including phenomena such as bulk flow, chemical reactions, phase equilibria, and heat and mass transfer. The unit model also contains performance equations which describe the behavior of these interaction phenomena, plus a number of *Ports* representing material flows into and out of the equipment. Finally, a flowsheet model describes an entire process (or sub-process) consisting of multiple unit models connected using *Arcs*, which are Pyomo components that link pairs of *Ports* using a set of equality constraints.

For each part of the modeling hierarchy, new types of modeling components have been developed to automate many of the common tasks associated with creating process models and to provide tools to perform common activities. These components are based on Pyomo *Blocks* and each will be described briefly in the following sections. Further details and examples may be found in the Supplemental Material and in the IDAES documentation (https://idaes-pse.readthedocs.io/en/stable/). Due to the hierarchical nature of the modeling framework, higher

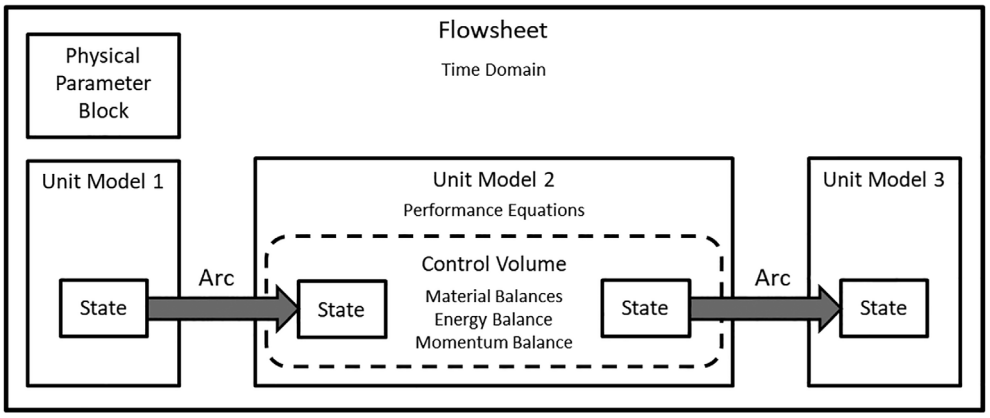

FIGURE 2 IDAES modeling hierarchy

level components depend on concepts introduced in lower level components; thus, these modeling components will be described from the bottom up.

3.3 | State and physical parameter blocks

The core of any process model is the set of calculations for determining the thermophysical properties of the material within the system. Accurate and efficient calculation of these properties is critical to being able to develop and solve a model for any process, and the formulation of these calculations has a significant impact on the overall tractability of the problem.

A large number of software packages exist which offer calculations of a wide range of thermophysical properties with access to large databanks or parameters regressed from experimental data. However, most of these packages are procedural in nature and operate as black-box models, which limits their integration within an EO environment. While Pyomo provides an interface for including such external function calls within models, this interface requires the black-box model to provide full first and second partial derivative information which is often not available, and not all solvers support the use of external functions. Future work will look at expanding the capabilities of the external function interface and interfacing with existing thermophysical property packages.

To provide calculation of thermophysical property within the IDAES-CMF, the state variables and constraints required to calculate the necessary thermophysical properties are implemented directly within the overall model through two modeling components: *State Blocks* and *Physical Parameter Blocks*.

State Blocks are the most commonly used component in the framework and are used where thermophysical property calculations are required. Each *State Block* is a self-contained sub-model representing the state of a given material at a point in space and time containing a set of variables describing the material state, plus additional variables, expressions, and constraints describing each property of interest at that point. As such, these represent the "states" shown in Figure 2. The thermophysical property calculations within these *State Blocks* depend on a set of parameters or physical constants which are constant for a given material. Duplicating these constants in each *State Block* would be inefficient and make it difficult to include these in a parameter estimation study. Thus, all constants for a given material are collected in a *Physical Parameter Block* which acts as a central repository of information for that material.

Given the highly non-linear nature of typical correlations for thermophysical properties, it can often be beneficial for convergence to represent the material state in different forms. For example, sometimes total flows and mole fractions will prove to be more reliable, while in other applications component flowrates may prove to be a better choice. The IDAES-CMF provides the flexibility to choose between any set of state variables, rather than enforcing a standard form, allowing for the most tractable set of state variables to be chosen for a given set of thermophysical property correlations. Each set of thermophysical property calculations is contained within a separate *State Block*, enabling the use of different sets of state variables in different parts of the flowsheet. Additionally, rather than requiring all possible thermophysical properties to be defined within a model, the IDAES-CMF requires only those properties used within a particular flowsheet to be defined. This greatly simplifies the construction of new thermophysical property models by reducing the number of properties that need to be defined before the model can be used in a flowsheet.

Examples of a *Physical Parameter Block* and a *State Block* definition can be found in the Supplementary Material.

3.4 | Reaction and reaction parameter blocks

Many processes also involve chemical reactions; however, these tend to be isolated to specific unit operations within the process, unlike thermophysical properties which are required in all unit operations. Thus, chemical reactions are treated separately from thermophysical properties, which allows reactions to be defined only for those unit operations where they are important and to allow different sets of reactions in different unit operations. In the same way that thermodynamic property calculations are modeled using *State Blocks* and *Physical Parameter Block* components, chemical reactions are modeled using *Reaction Block* and *Reaction Parameter Block* components. For the purposes of the modeling framework, two types of chemical reactions are considered; equilibrium reactions which depend solely on the material state, and rate- (kinetic) or yield-based reactions which also depend on the unit operation (e.g. reactor volume or a fixed conversion).

3.5 | Homogenous control volumes

While every process unit operation is different, they all share several common components, that is, material states which are linked together by a system of material, energy and momentum balances. The IDAES-CMF automates the creation of these common components, ultimately speeding up the model development process compared to process modeling in other AMLs while also eliminating one of the most common sources of modeling errors.

To do this, the IDAES-CMF uses control volumes which represent distinct volumes of material over which designated state transformations will occur. In the simplest form, a control volume has two material states representing the state of material entering and leaving the control volume, which are linked by a set of material, energy and momentum balances. To accelerate the development of new unit models, the IDEAS-CMF contains *Control Volume* components which contain a library of methods to automate common activities, such as the creation of *State Blocks* and writing material, energy and momentum equations.

Homogeneous *Control Volumes* support several different forms for the material (Equations (1)–(3)), energy (Equation (4)) and momentum balances (Equation (5)), since one form may be more tractable than others for a given application. More forms will be added in the future.

Here V is the volume of the control volume, ϕ_p is the volume fraction of phase p, I is the set of components, P is the set of thermodynamic phases of interest, E is the set of elements, R is the set of all reactions of interest (including rate-based, chemical equilibrium and phase equilibrium reactions), $C_{p,i}$ is the concentration of component i in phase p within the control volume, $F_{p,i,in}$ and $F_{p,i,out}$ are the flow of component i in phase p into and out of the control volume, $\nu_{e,i}$ is the moles of element e per mole of component i, $\alpha_{p,i,r}$ is the stoichiometric coefficient for component i in phase p in reaction r,X_r is the extent of reaction r and M is a generic term for other mass transfer phenomena crossing the control volume boundary. The component and phase-component forms support both mass and mole bases, with appropriate conversions where required (only molar basis is supported for element balances).

Basis	Form	
Component	$\sum_{p}^{P}\frac{\partial(VC_{p,i})}{\partial t}=\sum_{p}^{P}F_{p,i,in}-\sum_{p}^{P}F_{p,i,out}+\sum_{p}^{P}\sum_{r}^{R}\alpha_{p,i,r}X_r+M_i \quad \forall i\in I$	*Eqn 1*
Phase-Component	$\frac{\partial(\phi_p VC_{p,i})}{\partial t}=F_{p,i,in}-F_{p,i,out}+\sum_{r}^{R}\alpha_{p,i,r}X_r+M_{p.i} \quad \forall(p,i)\in I\times P$	*Eqn 2*
Element (mole basis only)	$\sum_{(p,i)}^{I\times P}\frac{\partial(V\nu_{e,i}C_{p,i})}{\partial t}=\sum_{(p,i)}^{I\times P}\nu_{e,i}F_{p,i,in}-\sum_{(p,i)}^{I\times P}\nu_{e,i}F_{p,i,out}+M_e \quad \forall e\in E$	*Eqn 3*

Basis	Form	
Total Enthalpy	$\sum_{p}^{P}\frac{\partial(V\hat{E}_p)}{\partial t}=\sum_{p}^{P}H_{p,in}-\sum_{p}^{P}H_{p,out}+Q+W+\Delta H_{MT}\left[+\sum_{r}^{R}X_r\Delta H_{rxn,r}\right]$	*Eqn 4*

Here $\hat{E}_p$ is the volume-specific internal energy of the material in phase p (i.e., the internal energy density), H_{in} and H_{out} are the enthalpy flow into and out of the control volume respectively, Q and W are heat and work transfer terms respectively, ΔH_{MT} is the energy change due to mass transfer, and $\Delta H_{rxn,r}$ is the molar heat of reaction for

TABLE 2 List of optional terms in balance equations

Material balances	Energy balances
Accumulation	Accumulation
Phase equilibrium[a]	Heat transfer
Rate-based chemical reactions[a]	Work transfer
Equilibrium-based chemical reactions[a]	Heat of reaction
Mass transfer	Energy change due to mass transfer
Momentum balances	
Pressure change	

[a]Not applicable to elemental balances.

reaction r. The $\Delta H_{rxn,r}$ term is included as an optional term to allow for short-cut approaches where heats of reaction are included explicitly in the energy balance equation rather than through heats of formation included in the enthalpy flow terms. Note that this is an either-or decision, and that defining enthalpy without the inclusion of heats of formation precludes the calculation of quantities such as Gibbs energy and thus the use of the Gibbs reactor model.

Basis	Form	
Total Pressure	$0 = P_{in} - P_{out} + \Delta P$	*Eqn 5*

Here P_{in} and P_{out} are the pressure at the inlet and outlet of the control volume, and ΔP is the pressure change between inlet and outlet.

These balance equations are all written in a general form and include terms for the common interactions that may occur; however, most applications will not require all of these terms. For example, a heating unit operation generally does not involve work transfer, thus this term is not required. Rather than including all terms within the balance equations and requiring values to be assigned to all of these, the IDAES-CMF allows for only those terms which are required in a given unit model to be identified and included in the constraints. Table 2 provides a list of all the optional terms for the different balance types; modelers are free to choose which terms from this list should be included in the balance equations of their models.

By providing a range of options with a simple interface, *Control Volumes* make it easy to study alternative formulations and find the most tractable form. Rather than having to edit or replace several complex constraints in the model, the form of the balance equations can be quickly changed by modifying one line of code, greatly reducing the time required to study different formulations and reducing the possibility of introducing errors. Additionally, optional terms can be readily added or removed to the balance equations allowing changes to add or remove complexity. This also supports the seamless transition between dynamic and steady-state models, as the time-dependent terms can be easily added or removed as required.

An example of the application of a homogeneous control volume to facilitate the construction of a model for a reactor can be found in the Supplementary Material.

3.6 | One-dimensional control volumes

While the simple inlet–outlet form described above is sufficient for many process applications, more complex unit operations may require consideration of spatial variances within the control volume. The IDAES-CMF provides support for control volumes with spatial variation in one dimension, and Equations (6) through (10) show the different forms available—note that due to spatial variations all transfer terms are defined in terms of transfer per unit length. Though standard control volumes do not yet exist for higher-dimensional systems, the IDAES-CMF has the necessary features to support any number of dimensions.

Basis	Form	
	Material Balances	
Component	$L\sum_{p}^{P}\frac{\partial\left(AC_{p,i,x}\right)}{\partial t} = \gamma\sum_{p}^{P}\frac{\partial F_{p,i,x}}{\partial x} + L\sum_{p}^{P}\sum_{r}^{R}\alpha_{p,i,r}X_{r,x} + LM_{i,x} \quad \forall i \in I$	*Eqn 6*
Phase-Component	$L\frac{\partial\left(\phi_p AC_{p,i,x}\right)}{\partial t} = \gamma\frac{\partial F_{p,i,x}}{\partial x} + L\sum_{r}^{R}\alpha_{p,i,r}X_{r,x} + LM_{p,i,x} \quad \forall (p,i) \in I \times P$	*Eqn 7*
Element (mole basis only)	$L\sum_{(p,i)}^{I\times P}\frac{\partial\left(A\nu_{e,i}C_{p,i,x}\right)}{\partial t} = \gamma\nu_{e,i}\sum_{(p,i)}^{I\times P}\frac{\partial F_{p,i,x}}{\partial x} + LM_{e,x} \quad \forall e \in E$	*Eqn 8*
	Energy Balances	
Total Enthalpy	$L\sum_{p}^{P}\frac{\partial\left(A\hat{E}_{p,x}\right)}{\partial t} = \gamma\sum_{p}^{P}\frac{\partial H_{p,x}}{\partial x} + LQ_x + LW_x\left[+L\sum_{r}^{R}X_{r,x}\Delta H_{rxn,r}\right]$	*Eqn 9*
	Momentum Balances	
Total Pressure	$0 = \gamma\frac{\partial P_x}{\partial x} + \Delta P_x$	*Eqn 10*

Here x is the normalized spatial domain, γ is a flow-direction term equal to -1 for flow from $x = 0$ to $x = 1$, L is the length of the spatial domain and A is the cross-sectional area of the control volume. For systems with pressure driven flow, additional constraints describing the ΔP_x term in Equation (10) as a function of velocity may be added, which can be calculated from the total material flowrate and density. Boundary conditions are specified by defining the state variables at the inlet to the control volume. The DAE toolbox in Pyomo[31] allows these equations to be written in their natural PDAE form and to be automatically discretized using several in-built schemes, which provides the ability to easily test different discretization schemes in order to determine the one most suitable for a given application without the need for modifying the model code.

One-dimensional *Control Volumes* support the same set of optional terms as homogeneous control volumes, allowing for simple addition and removal of terms as required. There are plans to include support for diffusion terms in future versions of the IDAES-CMF. For systems requiring models with more than one spatial dimension, the modeling framework supports higher-order derivatives however these would need to be custom coded. The IDAES-CMF also supports direct inclusion of surrogate or reduced-order models within the IDAES Integrated Platform.

3.7 | Unit model blocks

The components mentioned above provide the necessary building blocks for constructing unit operations models, which are described using *Unit Model* blocks. Each *Unit Model* block generally contains one or more *Control Volumes* describing the flow of material through the unit (however these can be written without control volumes), a set of performance constraints describing any undefined terms within the control volume balance equations (e.g. mass, heat and work transfer terms), and *Ports* describing the inlets and outlets from the unit.

Most of the complexity in a unit operation lies in defining the material, energy and momentum balances which are handled by the control volume(s). The unit model only needs to determine which forms to use for the balance equations and which terms should be included before calling the appropriate methods from the control volume, thus reducing the effort of writing these equations to three lines of code. Unit models may in turn include support for different forms through optional inputs during construction. As the *Unit Model* is "aware" of which terms will be included in the balance equations, it also needs to construct any performance constraints necessary to define these as appropriate (or leaving these as degrees of freedom to be defined when specifying design conditions).

Finally, unit models define the *Ports* which provide the connectivity between models within a flowsheet. The only information that needs to be passed between *Unit Models* is the state of the material at the inlet or outlet, which is fully described by the state variables in the associated *State Block*. Due to this, *Unit Models* can automate the construction of *Ports*, removing the need for them to be manually defined.

An example of defining a custom model for a reactor unit operation can be found in the Supplementary Material.

3.8 | Flowsheet blocks

The final component type provided by the IDAES-CMF is the *Flowsheet Block*, which serve as the canvas for constructing a process. Beginning with a *Flowsheet Block*, other types of IDAES modeling components are added to represent unit operations and thermophysical property calculations. The IDAES-CMF allows for *Flowsheet Blocks* to contain multiple *Physical Property Blocks* which can be used to represent streams of different materials, or to allow different methods of calculating properties for the same stream in different parts of the flowsheet. This feature can be used to help balance the need for accuracy versus tractability by limiting detailed but hard to solve property calculations to only those units where the accuracy has a significant impact on the model results.

Flowsheet Blocks also serve to organize and maintain representations of global phenomena, the most significant of which is the time domain for the process. To allow seamless transition between steady-state and dynamic modeling, all models in the IDAES-CMF contain a time domain which is used to index any property or variable which may vary with time. This is true even for steady-state cases where temporal variations are not of interest and the time domain consists of a single point. This is done to avoid structural differences (i.e. different indexing of variables and constraints) between dynamic and steady-state models so that the same general model can be applied to either application, allowing a seamless transition between steady-state and dynamic simulations.

An example of creating a flowsheet for a process will be demonstrated in the following case study.

3.9 | Initialization

When dealing with large-scale nonlinear systems, providing a good set of initial values for the variables in the model is often as important as constructing the model and choosing the right formulation. Any tool with a modular framework for assembling models of complex processes must also provide methods for these sub-models to reliably take them from an uninitialized state to one which can be solved for the conditions of interest. Commercial vendors have dedicated significant resources to this problem, and each has developed their own methods for doing this, which are generally proprietary.

While the IDAES-CMF is built around an EO paradigm, where the entire flowsheet is solved simultaneously, each unit operation is a self-contained model that can be solved in isolation from the flowsheet. This in-built hierarchical nature is a significant advantage over other AMLs, which generally view the model as a single set of constraints, because it allows easy identification of sub-models that can be solved in isolation in order to initialize the model. The IDAES-CMF takes a set of initial guesses for the state variables as inputs to automate initialization based on equation tearing and partitioning strategies for structured sub-models. All IDAES-CMF components (such as *State Blocks*, *Control Volumes* and *Unit Models*) are constructed with such *"initialize"* methods, which screen for inconsistent initial conditions and develop conditions that are well-posed for the system model. These *"initialize"* methods may call the *"initialize"* methods of other sub-models, providing a sequenced, hierarchical approach to develop a set of initial conditions.

To assist with development of initialization routines, Pyomo and IDAES have a number of supporting tools for initializing models, such as automated sequential decomposition routines, methods for easily copying state information between *State Blocks*, and integrators for dynamic problems.

4 | CASE STUDY—HDA PROCESS

To demonstrate the application of the IDAES-CMF, a case study is presented which shows how the tools can be used to build a flowsheet, which is then solved as a simulation. This is followed by an example of deterministic optimization to minimize total annualized cost. Finally, stochastic optimization under uncertainty is demonstrated on the same problem. A simplified flowsheet for the hydrodealkylation (HDA) of toluene to form benzene will be used for this case study based on the 1967 AIChE Student Contest problem as presented by Douglas,[32] which is available as an example in the IDAES software release.[1]

Hydrodealkylation is a chemical reaction that involves reacting an aromatic hydrocarbon in the presence of hydrogen gas to form a simpler aromatic hydrocarbon devoid of functional groups. In this example, toluene is reacted with hydrogen gas at high temperatures to form benzene via the following reaction:

$$C_6H_5CH_3 + H_2 \rightleftharpoons C_6H_6 + CH_4 \quad (11)$$

This reaction is often accompanied by an equilibrium side reaction which forms diphenyl, which is neglected for this example.

Figure 3 shows the process flow diagram for this case study, where 99.5 kg/hr (0.3 mol/s) of pure toluene is mixed with an equimolar flow of hydrogen with a ~ 6.7% methane impurity at 350 kPa and 303.15 K along with a recycle stream from the reactor (M101). The mixed stream is then fed to a pre-heater (H101) where it is heated to 600 K before passing to the reactor (R101). The reactor products are then fed to a flash unit (F101) where the stream is cooled to 325 K to separate the light gases (hydrogen and methane) from the benzene and toluene. The vapor stream is then sent to a separator (S101) where 20% of the light gases, along with any uncondensed benzene and toluene, are purged to remove CH_4 from the system, and the remainder recycled to the feed mixer via compressor P101. The liquid product stream from the flash separator is sent to a pre-heater unit (H102) before entering a distillation column (C101) to separate the benzene product from the remaining toluene.

4.1 | Step 1: Create the flowsheet

The first step in setting up the problem is to import Pyomo and the *Flowsheet Block* object, and to create a flowsheet as shown in Figure 4. To provide a basis for the problem and provide access to Pyomo's libraries and tools, a Pyomo model must first be created (line 8), to which the *Flowsheet Block* is attached (line 11).

4.2 | Step 2: Define the thermophysical and reaction properties

The next step is to add the required thermophysical and reaction property packages. For this case study, thermophysical properties for benzene, toluene, hydrogen and methane are required, along with a calculation of the rate of the hydrodealkylation reaction. Calculations for specific enthalpy (required for energy balances)

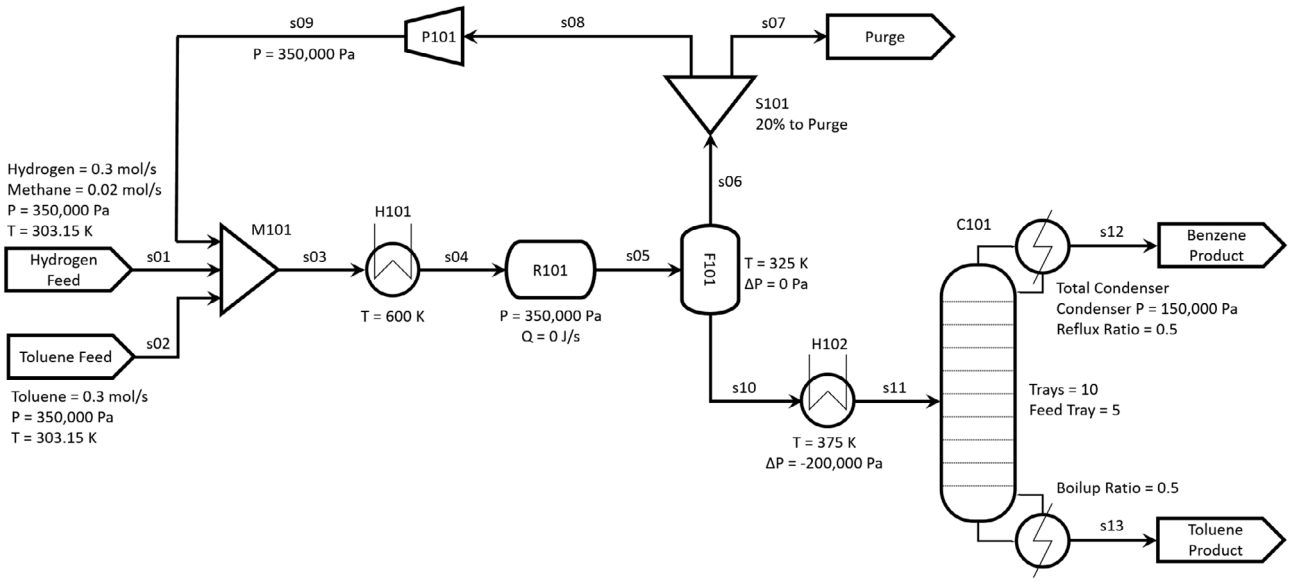

FIGURE 3 HDA process flow diagram

```
# Import Pyomo
import pyomo.environ as pyo

# Import IDAES FlowsheetBlock
from idaes.core import FlowsheetBlock

# Create a Pyomo Concrete Model to contain problem
m = pyo.ConcreteModel()

# Add a steady-state Flowsheet Block to model
m.fs = FlowsheetBlock(default={"dynamic": False})

# Import property packages
from idaes.generic_models.properties.activity_coeff_models.\
    BTX_activity_coeff_VLE import BTXParameterBlock
import hda_reaction_kinetic as reaction_props
from hda_ideal_VLE import HDAParameterBlock

# Add property parameter blocks to flowsheet
m.fs.bthm_params = HDAParameterBlock()
m.fs.bt_params = BTXParameterBlock(default={
    "activity_coeff_model": "Ideal"})
m.fs.reaction_params = reaction_props.HDAReactionParameterBlock(
    default={"property_package": m.fs.bthm_params})
```

FIGURE 4 Creating a flowsheet and adding property packages

and phase equilibrium are used in all of the unit operations. For the purposes of this case study, the following assumptions were made:

- Ideal gas behavior
- Reference state is 101 325 Pa and 298.15 K
- Phase equilibrium of all components follows Raoult's Law
- Smooth vapor–liquid equilibrium formulation[33]
- Gas phase molar enthalpies, heats of vaporization, and saturation pressures are from Reid et al.[34]
- Saturation pressures calculated using Antoine equation with parameters regressed from data in Reid et al.[34]
- Liquid phase molar densities and enthalpies as reported by Perry et al.,[35] 7th Edition, pages 2–94 and 2–170

Since benzene and toluene are separated by distillation, the HDA process inherently involves phase equilibrium and two-phase flow conditions. However, this is complicated by the presence of hydrogen and methane in much of the process, which under the conditions of interest are essentially non-condensable; that is, a vapor phase will be present under all conditions of interest and the bubble point of the mixture is extremely low. To simplify the phase-equilibrium calculations for this problem, hydrogen and methane will be considered completely non-condensable and insoluble in the liquid phase. Thus, no phase equilibrium constraint will be written for these species and their liquid phase compositions and flow rates will be set to a very small positive number (to avoid potential numerical issues with zero flows). EO solvers have numerous ways to handle small values and poorly conditioned systems, such as through scaling and regularization, which allow them to deal with these situations, whereas zero flows may cause singularities which result in failures when evaluating the model constraints.

Due to these assumptions the liquid stream leaving the flash separator (S101), and the distillation column and its pre-heater, will be completely free of hydrogen and methane. Since no hydrogen and methane will be present in these unit operations, a different component list can be used to simplify property calculations. The IDAES-CMF supports this by allowing the definition of multiple property packages within a single flowsheet and the use of "Translator" blocks to convert between different property calculations, component lists, and equations of state to provide full control over the formulation and rigor of different parts of the flowsheet.

To demonstrate this, two separate property packages will be used in the flowsheet; one for the reactor loop which contains all four species, and a second simpler set of calculations for only benzene and toluene following the flash separator. Additionally, the four-component property package uses molar flow indexed by phase and components (e.g., the molar flow of hydrogen in the vapor phase), temperature and pressure, while the two-component property package uses total flow and overall mole fractions in order to demonstrate the ability to change state variables as well.

For the reaction kinetics, a simplified rate expression based on that proposed by Douglas[32] will be used as shown in Equation (12), where r is the rate of reaction expressed in units of mol/m^3·s, $A = 6.3 \times 10^{10}$ mol/m^3·s·Pa is the Arrhenius pre-exponential factor, $E_A = 217.6$ kJ/mol·K is the activation energy for the reaction and p_T is the partial pressure of toluene.

$$r = Ae^{-E_A/RT}p_T \tag{12}$$

Additionally, the reaction package needs to define the stoichiometry of the HDA reaction and the associated heat of reaction (as heat of formations are not included in the calculation of specific enthalpy for this example).

Due to space constraints, the full code of the property packages will not be shown here. A simple example of a physical property package (a physical parameter block and state block) is shown in the Supplementary Material, and full examples of the property packages can be found in the IDAES public release.[1] The full set of thermophysical property constraints, including those for phase equilibrium, are implemented directly within the problem structure (rather than as black-box external function calls). Thus, the numerical solver is provided with exact first and second partial derivatives for all the property calculations. This allows for much greater solution efficiency without requiring multiple solver calls to approximate the gradient using finite difference-based approaches.

Line 20 in Figure 4 shows how the four-component property package is added to the flowsheet (named *bthm_params*), followed by the two-component property package (*bt_params*) on Lines 21 and 22. Finally, the reaction package is added to the flowsheet on Lines 23 and 24, including linking it with the four-component property package used in the reactor unit.

4.3 | Step 3: Define the unit operations

Import Unit Operations from the IDAES Model Library: For this case study, the IDAES model libraries contain all the necessary unit operations which must be imported as shown in Figure 5—an example of creating a custom unit model can be found in the Supplementary Material. As the IDAES unit model library provides a set of generic unit operation models that cover a wide range of needs, most unit models support a number of options, such as the number and names of inlets, or whether to include a pressure drop across a unit. For some options, there is a set of predefined choices which are implemented as enumerated objects, which must also be imported. Examples of these can be seen on lines 33 to 36 of Figure 5.

Defining the Feed Objects: The first stage in any flowsheet is to define the feed streams entering the process using IDAES Feed blocks. This process has two feed streams, one for hydrogen and one for toluene, and lines 39 to 43 of Figure 5 show how these are defined in the flowsheet. Each unit model in the flowsheet must be linked to one or more property packages which will be used in the unit operation. This is done by providing a

```
25  # Import unit models from IDAES libraries
26  from idaes.generic_models.unit_models import (
27      CSTR, Feed, Flash, Heater, Mixer,
28      PressureChanger, Separator as Splitter, Translator)
29  from idaes.generic_models.unit_models.distillation import \
30      TrayColumn
31
32  # Import supporting definition objects
33  from idaes.generic_models.unit_models.distillation.condenser \
34      import CondenserType, TemperatureSpec
35  from idaes.generic_models.unit_models.pressure_changer \
36      import ThermodynamicAssumption
37
38  # Add Feed blocks to flowsheet
39  m.fs.hydrogen_feed = Feed(default={
40      "property_package": m.fs.bthm_params})
41
42  m.fs.toluene_feed = Feed(default={
43      "property_package": m.fs.bthm_params})
44
45  # Add a Mixer to the flowsheet
46  m.fs.M101 = Mixer(default={
47      "property_package": m.fs.bthm_params,
48      "inlet_list": [
49          "toluene_feed", "hydrogen_feed", "vapor_recycle"]})
50
51  # Add reactor pre-heater
52  m.fs.H101 = Heater(default={
53      "property_package": m.fs.bthm_params,
54      "has_pressure_change": False,
55      "has_phase_equilibrium": True})
56
57  # Add reactor
58  m.fs.R101 = CSTR(default={
59      "property_package": m.fs.bthm_params,
60      "reaction_package": m.fs.reaction_params,
61      "has_heat_of_reaction": True,
62      "has_heat_transfer": True,
63      "has_pressure_change": False})
```

FIGURE 5 Adding feed blocks, mixer, heater, and reactor

pointer to the instance of the Physical Parameter Block previously attached to the flowsheet as shown in lines 40 and 43.

Defining the Feed Mixer (M101): The first unit operation in the HDA process is the feed mixer (M101) which combines the two feed streams plus the recycle stream, which can be modeled using the default Mixer unit model from the IDAES model library. In this case, two options are required which are shown in Figure 5, lines 46 to 49; in addition to the pointer to the appropriate property package, the inlets required for the mixer need to be defined, which is done using the *"inlet_list"* option (lines 48 and 49).

Defining the Feed Pre-Heater (H101): The next unit operation in the process is the reactor pre-heater. For this example, pressure drop across the unit will be neglected (and thus excluded from the momentum balance calculations). However, the mixture may form two phases under some circumstances, thus phase equilibrium must be considered in the heater. Lines 52 to 55 in Figure 5 show the code for adding the pre-heater to the flowsheet with the required options.

Defining the Reactor (R101): In this demonstration, the reactor will be modeled as a CSTR (lines 58 to 63 in Figure 5). In addition to a thermodynamic property package, reactors require a reaction property package to specify properties related to reactions such as stoichiometry and heats of reaction (line 60). Whether heat of reaction needs to be explicitly included in the energy balances needs to be specified (line 61, it can also

```
64  # Add flash separator
65  m.fs.F101 = Flash(default={
66      "property_package": m.fs.bthm_params,
67      "has_heat_transfer": True,
68      "has_pressure_change": True})
69
70  # Add purge split
71  m.fs.S101 = Splitter(default={
72      "property_package": m.fs.bthm_params,
73      "ideal_separation": False,
74      "outlet_list": ["purge", "recycle"]})
75
76  # Add recycle compressor
77  m.fs.P101 = PressureChanger(default={
78      "property_package": m.fs.bthm_params,
79      "compressor": True,
80      "thermodynamic_assumption":
81          ThermodynamicAssumption.isothermal})
```

FIGURE 6 Adding flash separator, purge splitter, and recycle compressor

be implicitly included via the component specific enthalpies defined in the thermodynamic property package by including heats of formation), as well as whether the reactor should incorporate a heat duty term (line 62).

Defining the Flash Drum (F101), Purge Splitter (S101), and Recycle Compressor (P101): Figure 6 shows how to add the next set of unit operations for light gas separation and to set up the recycle loop in the flowsheet. In the case of the recycle compressor (P101), it is necessary to define whether the unit is a compressor or expander (*"compressor"* option, line 79) (which determines how efficiency should be included in determining the mechanical work for the unit), and what thermodynamic assumption should be used when calculating work (lines 80 and 81). For this case study, the compressor work is not important (the compressor work will not be considered in the objective function) and a trivial isothermal pressure change will be used to minimize the complexity of the resulting equations (as a result, efficiency is not considered either).

Adding the Translator Block: As discussed in Step 2, a Translator block is required after the flash separator to switch from the four-component property package to the two-component property package, and lines 83 to 85 in Figure 7 show how to add a Translator block to the flowsheet. In this case, it is necessary to link two property packages to the Translator block—the source property package and the package that it is to be translated to. Additionally, the Translator block needs to know how to translate between the two property packages; due to the generality of the IDAES-CMF this must be custom coded for each application. In this case, five constraints are required based on the state variables used in the outgoing property package:

1. Total molar flowrate (lines 89 to 94): in this case, it is assumed that only liquid benzene and toluene will actually be present in the stream, thus the total component flowrate is the sum of the liquid phase flowrates of benzene and toluene in the incoming stream.
2. Temperature equality (lines 97 to 99): while an energy balance might seem like a better choice here, enthalpy (or internal energy) is a relative quantity and is dependent on the reference state and correlations used in each property package. Thus, enthalpy cannot (in general) be directly equated between different property packages. Temperature is an absolute quantity however, and thus independent of the reference state or property correlations and can therefore be directly equated when translating between any two property packages.
3. Pressure equality (lines 102 to 104)
4. Overall mole fraction of benzene (lines 107 to 115): the overall mole fraction of benzene in the outgoing stream will be equal to the incoming flowrate of liquid benzene divided by the sum of the incoming flowrates of liquid benzene and toluene.
5. Overall mole fraction of toluene (lines 117 to 125): similar to the above constraint for benzene.

Defining the final pre-heater (H102) and distillation column (C101): The remaining units in the flowsheet

```
82  # Add translator block to convert between property packages
83  m.fs.translator = Translator(default={
84      "inlet_property_package": m.fs.bthm_params,
85      "outlet_property_package": m.fs.bt_params})
86
87  # Add constraints to link inlet and outlet states
88  # Total flow constraint
89  m.fs.translator.eq_total_flow = pyo.Constraint(
90      expr=m.fs.translator.outlet.flow_mol[0] ==
91      m.fs.translator.inlet.flow_mol_phase_comp[
92          0, "Liq", "benzene"] +
93      m.fs.translator.inlet.flow_mol_phase_comp[
94          0, "Liq", "toluene"])
95
96  # Temperature equality
97  m.fs.translator.eq_temperature = pyo.Constraint(
98      expr=m.fs.translator.outlet.temperature[0] ==
99      m.fs.translator.inlet.temperature[0])
100
101 # Pressure equality
102 m.fs.translator.eq_pressure = pyo.Constraint(
103     expr=m.fs.translator.outlet.pressure[0] ==
104     m.fs.translator.inlet.pressure[0])
105
106 # Total mole fractions of benzene and toluene
107 m.fs.translator.eq_mole_frac_benzene = pyo.Constraint(
108     expr=m.fs.translator.outlet.mole_frac_comp[
109         0, "benzene"] ==
110     m.fs.translator.inlet.flow_mol_phase_comp[
111         0, "Liq", "benzene"] /
112     (m.fs.translator.inlet.flow_mol_phase_comp[
113         0, "Liq", "benzene"] +
114      m.fs.translator.inlet.flow_mol_phase_comp[
115          0, "Liq", "toluene"]))
116
117 m.fs.translator.eq_mole_frac_toluene = pyo.Constraint(
118     expr=m.fs.translator.outlet.mole_frac_comp[
119         0, "toluene"] ==
120     m.fs.translator.inlet.flow_mol_phase_comp[
121         0, "Liq", "toluene"] /
122     (m.fs.translator.inlet.flow_mol_phase_comp[
123         0, "Liq", "benzene"] +
124      m.fs.translator.inlet.flow_mol_phase_comp[
125         0, "Liq", "toluene"]))
```

FIGURE 7 Adding translator block and associated constraints

are the second pre-heater and the distillation column. Figure 8 shows how these are added to the flowsheet. Note that both of these units use the two-component property package (lines 128 and 140) rather than the four-component property packages used by the other unit models. In the case of the distillation column, there are a number of options which need to be defined, such as number of trays and location of the feed tray (lines 134 and 135), the condenser type and specification (lines 136 to 139) and whether the column will be isobaric (line 142).

4.4 | Step 4: Define the flowsheet connectivity

Once all the unit operations have been added to the flowsheet, the next step is to define connectivity between the units. This is done using the *Arc* component from the *pyomo.network* toolbox which is imported as shown in Figure 9, line 144; the *Sequential Decomposition* tool will be utilized later. *Arcs* represent the streams within the process and are defined by linking the outlet *Port* of one unit model to the inlet *Port* of the next, as shown in

```
126 # Add distillation pre-heater
127 m.fs.H102 = Heater(default={
128     "property_package": m.fs.bt_params,
129     "has_pressure_change": True,
130     "has_phase_equilibrium": True})
131
132 # Add distillation column
133 m.fs.C101 = TrayColumn(default={
134     "number_of_trays": 10,
135     "feed_tray_location": 5,
136     "condenser_type":
137         CondenserType.totalCondenser,
138     "condenser_temperature_spec":
139         TemperatureSpec.atBubblePoint,
140     "property_package": m.fs.bt_params,
141     "has_heat_transfer": False,
142     "has_pressure_change": False})
```

FIGURE 8 Adding distillation pre-heater and tray column

```
143 # Import Pyomo network tools
144 from pyomo.network import Arc, SequentialDecomposition
145
146 # Define streams connecting unit operations
147 m.fs.s01 = Arc(source=m.fs.toluene_feed.outlet,
148                destination=m.fs.M101.toluene_feed)
149 m.fs.s02 = Arc(source=m.fs.hydrogen_feed.outlet,
150                destination=m.fs.M101.hydrogen_feed)
151 m.fs.s03 = Arc(source=m.fs.M101.outlet,
152                destination=m.fs.H101.inlet)
153 m.fs.s04 = Arc(source=m.fs.H101.outlet,
154                destination=m.fs.R101.inlet)
155 m.fs.s05 = Arc(source=m.fs.R101.outlet,
156                destination=m.fs.F101.inlet)
157 m.fs.s06 = Arc(source=m.fs.F101.vap_outlet,
158                destination=m.fs.S101.inlet)
159 m.fs.s08 = Arc(source=m.fs.S101.recycle,
160                destination=m.fs.P101.inlet)
161 m.fs.s09 = Arc(source=m.fs.P101.outlet,
162                destination=m.fs.M101.vapor_recycle)
163 m.fs.s10a = Arc(source=m.fs.F101.liq_outlet,
164                 destination=m.fs.translator.inlet)
165 m.fs.s10b = Arc(source=m.fs.translator.outlet,
166                 destination=m.fs.H102.inlet)
167 m.fs.s11 = Arc(source=m.fs.H102.outlet,
168                destination=m.fs.C101.feed)
169
170 # Expand Arcs into equality constraints
171 pyo.TransformationFactory("network.expand_arcs").apply_to(m)
```

FIGURE 9 Connecting unit models in flowsheet using Pyomo arcs

Figure 9. Once all *Arcs* in the flowsheet have been defined, the Pyomo *Transformation Factory* is used, as shown in line 171 of Figure 9, to automatically expand the *Arcs* into sets of equality constraints linking the variables in the inlet and outlet ports of connected units.

4.5 | Step 5: Define custom calculations

In many cases, there may be some additional quantities of interest which may need to be defined. For example, the conversion in the reactor is often of interest when

```
172 # Define a variable for reactor conversion
173 m.fs.R101.conversion = pyo.Var(initialize=0.75, bounds=(0, 1))
174
175 # Add a constraint describing conversion
176 m.fs.R101.conv_constraint = pyo.Constraint(
177     expr=m.fs.R101.conversion*m.fs.R101.inlet.
178     flow_mol_phase_comp[0, "Vap", "toluene"] ==
179     (m.fs.R101.inlet.flow_mol_phase_comp[0, "Vap", "toluene"] -
180       m.fs.R101.outlet.flow_mol_phase_comp[0, "Vap", "toluene"]))
181
182 # Add expression for operating costs
183 m.fs.cooling_cost = pyo.Expression(
184     expr=0.25e-7 * (-m.fs.F101.heat_duty[0]) +
185     0.2e-7 * (-m.fs.C101.condenser.heat_duty[0]))
186 m.fs.heating_cost = pyo.Expression(
187     expr=2.2e-7 * m.fs.H101.heat_duty[0] +
188     1.2e-7 * m.fs.H102.heat_duty[0] +
189     1.9e-7 * m.fs.C101.reboiler.heat_duty[0])
190 m.fs.operating_cost = pyo.Expression(
191     expr=(3600*24*365*(m.fs.heating_cost + m.fs.cooling_cost)))
192
193 # Add expresion for capital cost
194 m.fs.capital_cost = pyo.Expression(expr=1e5*m.fs.R101.volume[0])
```

FIGURE 10 Adding additional variables, expression and constraints

examining a flowsheet; however, this is not defined as part of the core CSTR model as it is difficult to determine which is the key component that conversion should be based on without user input. Thus, it is necessary to define a variable and constraint as shown in Figure 10, lines 173 to 180.

For the optimization problem, expressions to compute operating costs due to heating and cooling utilities as well as the capital cost of the reactor unit will be added to the flowsheet. For this case study, the following costs are assumed:

- \$0.25e−7/J for the cooling utility in the flash separator (F101)
- \$0.2e−7/J for the cooling utility in the distillation condenser
- \$2.2e−7/J for the heating utility in the reactor preheater (H101)
- \$1.2e−7/J for the heating utility in the distillation preheater (H102)
- \$1.9e−7/J for heating utility in the distillation reboiler
- \$1e5/m^3·yr. for the capital cost of the reactor (R101), amortized over the lifetime of the process.

The different heating utility costs are based on the pressure/temperature of the steam used for each unit operation. Lines 183 to 186 of Figure 10 show the addition of an expression for the cost of cooling utilities, lines 187 to 189 show an expression for the heating utilities and lines 190 to 191 an expression for the overall utility cost for the process. Line 194 shows the addition of an expression for the amortized capital cost of the reactor.

4.6 | Step 6: Specify design conditions and verify the degrees of freedom

Now that the unit operations and connectivity of the flowsheet have been defined, the next step is to set the design and operating conditions of the process; that is, satisfying the degrees of freedom of the problem. The IDAES-CMF includes a degrees of freedom tool, which can be used as shown in lines 196 and 199 of Figure 11 to determine how many degrees of freedom the problem has. As can be seen, the current flowsheet has 29 degrees of freedom (line 200) which need to be specified.

The first 20 degrees of freedom that need to be specified are the feed conditions for the toluene and hydrogen streams (10 variables each). The feed conditions can be set through the Feed blocks as shown in Figure 11. Some important things to note when setting the feed conditions:

- A time index must be specified when fixing each variable. For dynamic simulations, Pyomo supports setting all time points simultaneously using slicer notation.
- To avoid convergence issues (e.g., singularities and degeneracy) associated with values of 0 for variables in EO models, all zero flows in this example will be fixed to a small positive value (i.e. 1e−8). This assumption

```
# Import IDAES degrees of freedom tool
from idaes.core.util.model_statistics import degrees_of_freedom

# Print degrees of freedom in model
print(degrees_of_freedom(m))
>>> 29

# Set toluene feed conditions
t_feed = m.fs.toluene_feed
t_feed.flow_mol_phase_comp[0, "Vap", "benzene"].fix(1e-8)
t_feed.flow_mol_phase_comp[0, "Vap", "toluene"].fix(1e-8)
t_feed.flow_mol_phase_comp[0, "Vap", "hydrogen"].fix(1e-8)
t_feed.flow_mol_phase_comp[0, "Vap", "methane"].fix(1e-8)
t_feed.flow_mol_phase_comp[0, "Liq", "benzene"].fix(1e-8)
t_feed.flow_mol_phase_comp[0, "Liq", "toluene"].fix(0.30)
t_feed.flow_mol_phase_comp[0, "Liq", "hydrogen"].fix(1e-8)
t_feed.flow_mol_phase_comp[0, "Liq", "methane"].fix(1e-8)
t_feed.temperature.fix(303.2)
t_feed.pressure.fix(350000)

# Set hydrogen feed conditions
h_feed = m.fs.hydrogen_feed
h_feed.flow_mol_phase_comp[0, "Vap", "benzene"].fix(1e-8)
h_feed.flow_mol_phase_comp[0, "Vap", "toluene"].fix(1e-8)
h_feed.flow_mol_phase_comp[0, "Vap", "hydrogen"].fix(0.30)
h_feed.flow_mol_phase_comp[0, "Vap", "methane"].fix(0.02)
h_feed.flow_mol_phase_comp[0, "Liq", "benzene"].fix(1e-8)
h_feed.flow_mol_phase_comp[0, "Liq", "toluene"].fix(1e-8)
h_feed.flow_mol_phase_comp[0, "Liq", "hydrogen"].fix(1e-8)
h_feed.flow_mol_phase_comp[0, "Liq", "methane"].fix(1e-8)
h_feed.temperature.fix(303.2)
h_feed.pressure.fix(350000)
```

FIGURE 11 Checking degrees of freedom and setting feed conditions

was chosen specifically for this case and is not appropriate or necessary for all cases, e.g., systems with trace elements or a different formulation for phase equilibrium. The value of 1e−8 was chosen because it is equal to the solver tolerance used (both absolute and relative) and is significantly smaller than the real flows in the system.

The remaining 12 degrees of freedom are the operating conditions for the unit operations, and the most appropriate design conditions for the application can be selected. For example, in a CSTR it is possible to specify the reactor volume and to solve for conversion, or to specify conversion and solve for the reactor volume required to achieve that conversion. The IDAES-CMF documentation provides users with some guidance on typical degrees of freedom to consider fixing, and future work is planned to develop tools for automatically identifying potential degrees of freedom to fix as well as degeneracies caused by poor selection of degrees of freedom. A summary of the design conditions that will be specified for each unit operation is given below:

- Reactor pre-heater (H101)—outlet temperature = 600 K
- Reactor (R101)—conversion = 75%, heat duty = 0 J/s
- Flash separator (F101)—vapor outlet temperature = 325 K, pressure change = 0 Pa
- Splitter (S101)—purge split fraction = 20%
- Recycle compressor (P101)—outlet pressure = 350 000 Pa (equal to feed pressures)
- Distillation pre-heater (H102)—outlet temperature = 375 K, pressure drop = 200 000 Pa
- Distillation column (C101)—reflux ratio = 0.5, condenser pressure = 150 000 Pa, boilup ratio = 0.5

Figure 12 shows how to set the design conditions for all units in the flowsheet.

4.7 | Step 7: Initialize the flowsheet using sequential decomposition

Once the degrees of freedom have been satisfied (the degrees of freedom tool can be called again to confirm), the model is ready to be initialized. As discussed

```
227 # Set reactor pre-heater operating conditions
228 m.fs.H101.outlet.temperature.fix(600)
229
230 # Set reactor conversion and heat duty
231 m.fs.R101.conversion.fix(0.75)
232 m.fs.R101.heat_duty.fix(0)
233
234 # Set flash separator operating conditions
235 m.fs.F101.vap_outlet.temperature.fix(325.0)
236 m.fs.F101.deltaP.fix(0)
237
238 # Set purge split and recycle conditions
239 m.fs.S101.split_fraction[0, "purge"].fix(0.2)
240 m.fs.P101.outlet.pressure.fix(350000)
241
242 # Set distillation pre-heater outlet conditions
243 m.fs.H102.outlet.temperature.fix(375)
244 m.fs.H102.deltaP.fix(-200000)
245
246 # Set distillation column operating conditions
247 m.fs.C101.condenser.reflux_ratio.fix(0.5)
248 m.fs.C101.condenser.condenser_pressure.fix(150000)
249 m.fs.C101.reboiler.boilup_ratio.fix(0.5)
```

FIGURE 12 Setting unit design conditions

previously, the IDAES-CMF supports a Sequential Decomposition (SD) approach when initializing a model, where each unit is initialized separately, propagating the outlet conditions from one unit to the inlet of the next. This process has a recycle stream, which requires tearing to determine where to start the SD initialization process. The *pyomo.network* toolbox contains an automated *Sequential Decomposition* tool[36] (imported earlier in Figure 9, line 144) which can both automatically determine tear streams and automate the SD initialization process. Note that the SD tool is only used for initialization and does not need to converge to the final solution for the overall model; it is only necessary to develop a sufficiently good initial guess for the EO solver to find the actual solution to the model.

Identifying the Tear Stream(s): Figure 13 shows how to set up the *Sequential Decomposition* tool and apply it to the flowsheet. First, a *Sequential Decomposition* object is created and the method to use for determining tear streams selected (lines 251–253). The *Sequential Decomposition* tool supports both heuristic based methods (*method*="*heuristic*," used here) and solving a mixed integer problem for finding tear streams (*method*="*mip*," which requires access to an appropriate mixed-integer solver). Additionally, a limit on the number of iterations when trying to converge tear streams while initializing the flowsheet can be set (line 256). As the flowsheet will later be solved in its entirety, the *Sequential Decomposition* tool does not need to fully converge the tear stream; hence, only 3 iterations will be used to sufficiently converge the tear stream for the EO solver to take over.

Once the *Sequential Decomposition* tool has determined the tears streams and solution order for the flowsheet (line 259 to 261 in Figure 13), these should be verified. An initial guess for the tear streams must be provided, so it is important to confirm the appropriateness of the tear stream(s), and the automated selection can be overridden when necessary. Lines 264 and 265 show how to check the tear streams selected by the *Sequential Decomposition* tool and the order in which the unit models will be initialized (lines 268 and 269).

Specify Initial Guess Values for the Tear Streams: For this case, the *Sequential Decomposition* tool identified the stream between the mixer and pre-heater (s03) as the tear stream for the process. Given the information available, it will be assumed that the tear stream consists of only the toluene and hydrogen feeds. Thus, the initial guesses for the tear stream can be calculated as the sum of these; however, to ensure robust solution of the phase-equilibrium constraints, the zero flowrates will be assigned an initial guess of 1e-5 for the SD initialization. The final solve of the flowsheet will use the values defined for the inlets. The initial guesses for the tear stream are set as shown in Figure 14, lines 271 to 282.

Perform Initialization using Sequential Decomposition Approach: Once the initial guesses for the tear stream are set, the next step is to set up instructions for the *Sequential Decomposition* tool to initialize each unit in the flowsheet. Each IDAES-CMF unit model comes

```
250 # Setup Pyomo Sequential Decomposition tool
251 seq = SequentialDecomposition()
252 seq.options.select_tear_method = "heuristic"
253 seq.options.tear_method = "Wegstein"
254
255 # Limit SD iterations
256 seq.options.iterLim = 3
257
258 # Use SD Tool to determine tear stream and initialization order
259 G = seq.create_graph(m)
260 heuristic_tear_set = seq.tear_set_arcs(G, method="heuristic")
261 order = seq.calculation_order(G)
262
263 # Print tear stream
264 for o in heuristic_tear_set:
265     print(o.name)
266
267 # Print initialization order
268 for o in order:
269     print(o[0].name)
```

FIGURE 13 Preparing the sequential decomposition tool

```
270 # Create a dictionary of guesses for tear stream
271 tear_guesses = {
272         "flow_mol_phase_comp": {
273                 (0, "Vap", "benzene"): 1e-5,
274                 (0, "Vap", "toluene"): 1e-5,
275                 (0, "Vap", "hydrogen"): 0.30,
276                 (0, "Vap", "methane"): 0.02,
277                 (0, "Liq", "benzene"): 1e-5,
278                 (0, "Liq", "toluene"): 0.30,
279                 (0, "Liq", "hydrogen"): 1e-5,
280                 (0, "Liq", "methane"): 1e-5},
281         "temperature": {0: 303},
282         "pressure": {0: 350000}}
283
284 # Pass the tear_guess to the SD tool
285 seq.set_guesses_for(m.fs.H101.inlet, tear_guesses)
286
287 # Define a method to initiaize unit operations
288 def function(unit):
289     unit.initialize(outlvl=100)
290
291 # Run the SD tool
292 seq.run(m, function)
293
294 # Create a solver object and solve the full flowsheet
295 solver = pyo.SolverFactory("ipopt")
296 solver.solve(m, tee=True)
```

FIGURE 14 Setting tear guesses and running sequential decomposition tool

with a pre-defined initialize method, and the *Sequential Decomposition* tool runs this for each unit in sequence, as shown in lines 288 and 289 in Figure 14. The *Sequential Decomposition* tool can then run the SD initialization procedure (line 292), which will iterate through the process and attempt to converge the tear stream(s) up to the iteration limit previously specified; it may terminate earlier if the tear streams converge. After the SD

TABLE 3 Stream table for deterministic flowsheet

Operating cost	$ 427 511/yr.				
Capital cost	$ 14 690/yr.				
Benzene purity	89.514%				
Conversion	75.00%				
Stream	s01	s02	s03	s04	s05
Flow rate (mol/s)					
Liq, benzene	1e−8	1e−8	2.8e−8	1.2986e−7	1.2985e−7
Liq, toluene	0.3	1e−8	0.30000	8.4136e−7	8.4136e−7
Liq, hydrogen	1e−8	1e−8	2.8e−8	1e−8	1e−8
Liq, methane	1e−8	1e−8	2.8e−8	1e−8	1e−8
Vap, benzene	1e−8	1e−8	0.11929	0.11929	0.35365
Vap, toluene	1e−8	1e−8	0.012487	0.31249	0.078122
Vap, hydrogen	1e−8	0.3	0.56254	0.56254	0.32818
Vap, methane	1e−8	0.02	1.03746	1.0375	1.2718
Temperature (K)	303.2	303.2	314.09	600.0	771.86
Pressure (Pa)	350 000	350 000	350 000	350 000	350 000
Stream	s06	s07	s08	s09	s10
Flow rate (mol/s)					
Liq, benzene	1e−8	1e−8	1e−8	1e−8	0.20454
Liq, toluene	1e−8	1e−8	1e−8	1e−8	0.062514
Liq, hydrogen	1e−8	1e−8	1e−8	1e−8	2.7e−7
Liq, methane	1e−8	1e−8	1e−8	1e−8	2.7e−7
Vap, benzene	0.14911	0.029822	0.11929	0.11929	1e−8
Vap, toluene	0.015609	3.1217e−3	0.012487	0.012487	1e−8
Vap, hydrogen	0.32818	0.065635	0.26254	0.26254	1e−8
Vap, methane	1.2718	0.25436	1.0175	1.0175	1e−8
Temperature (K)	325.0	325.0	325.0	325.0	325.0
Pressure (Pa)	350 000	350 000	350 000	350 000	350 000
Stream	s11	s12	s13		
Flow rate (mol/s)	0.26706	0.16196	0.10509		
Mole fraction					
Benzene	0.76592	0.89514	0.56676		
Toluene	0.23408	0.10486	0.43324		
Temperature (K)	375.00	368.93	377.30		
Pressure (Pa)	150 000	150 000	150 000		

initialization is completed, the flowsheet can be solved by calling an EO solver, in this case IPOPT[37] (lines 295 and 296, IPOPT version 3.12.2 with MA27).

The final size of the deterministic flowsheet was 1189 variables and constraints, and required 16.2 s for the SD initialization and 0.50 s (0.03 CPU seconds in IPOPT) for the final solve on a commercial laptop (Intel Core i7-8750H CPU @ 2.20 GHz, 10.4 GB memory).

Inspect the Initial Solution: Table 3 shows the results of the simulation in the form of a stream table. Due to the change of property package at stream 10, the state variables change as reflected on the stream table. The product stream flowrate (stream s12) is 0.162 mol/s with a benzene purity of 89.51%, and the reactor volume required to achieve a conversion of 75% is 0.147 m^3. However, stream s06 shows that a significant amount of

```
297 # Create an objective function for optimization
298 m.fs.objective = pyo.Objective(
299     expr=m.fs.operating_cost + m.fs.capital_cost)
300
301 # Add a constraint on benzene product flowrate
302 m.fs.product_flow = pyo.Constraint(
303     expr=m.fs.C101.condenser.distillate.flow_mol[0] >=
304     0.18)
305
306 # Add a constraint on benzene product purity
307 m.fs.product_purity = pyo.Constraint(
308     expr=m.fs.C101.condenser.
309     distillate.mole_frac_comp[0, "benzene"] >= 0.99)
310
311 # Add constraints on benzene loss
312 m.fs.overhead_loss = pyo.Constraint(
313     expr=m.fs.F101.vap_outlet.flow_mol_phase_comp[
314         0, "Vap", "benzene"] <=
315     0.20 * m.fs.R101.outlet.flow_mol_phase_comp[
316         0, "Vap", "benzene"])
```

FIGURE 15 Adding an objective function and inequality constraints

benzene (0.149 mol/s or approximately 42% of the total benzene produced in the reactor R101) is exiting the flash separator, which suggests that the operating conditions could be changed such that the loss of benzene in the flash separator is minimized.

For the simulation case, with the initial operating conditions that were set, the total annual cost (sum of amortized capital and operating cost) is $442 201.

The EO approach used by the IDAES-CMF allows for quickly switching from solving a simulation problem to solving an optimization problem. In this case study, the total annual cost of the HDA process will be minimized, while enforcing certain performance constraints (minimum production and purity requirement, and limit product loss), by adjusting the following 7 decision variables:

- Reactor pre-heater (H101) outlet temperature,
- reactor (R101) cooling duty,
- light gas separator (F101) outlet temperature,
- distillation pre-heater (H102) outlet temperature, and
- distillation column (C101) condenser pressure, reflux ratio and boilup ratio.

4.8 | Step 8: Define objective and remaining constraints

In addition to minimizing the total annualized cost, it is necessary to provide constraints on the final product, as well as a limit on the amount of benzene in the light gas stream leaving the flash separator. The following constraints are added to the process:

- produce at least 0.18 mol/s of benzene in C101 distillate outlet (i.e. the benzene product stream s12),
- purity of benzene product is at least 99% in the product stream s12, and
- no more than 20% of the benzene entering the flash separator (F101) may leave in the light gas stream s06.

The first step is to add an objective function to the model using the Pyomo *Objective* component as shown in Figure 15, Lines 298 and 299. The goal is to minimize the sum of operating and capital costs, which have already been added to the flowsheet as *Expressions*. Unless otherwise specified, Pyomo assumes the goal is to minimize the objective function. Figure 15 also shows how to add constraints to the model to enforce the product flow rate, purity and benzene loss conditions, which is done using inequality constraints.

4.9 | Step 9: Unfix the operating conditions to create degrees of freedom for optimization

Next, the operating conditions can be converted to decision variables simply by unfixing their values, as shown in Figure 16 (lines 318 to 324). However, it is also necessary to set bounds on these variables to limit the search space available to the solver (lines 327 to 341). For this case study, the following bounds will be used:

- H101 outlet temperature [500, 600] K
- R101 outlet temperature [600, 900] K
- R101 volume lower bound of 0 m^3 (no upper bound)

```
317 # Unfix decission variables
318 m.fs.H101.outlet.temperature.unfix()
319 m.fs.R101.conversion.unfix()
320 m.fs.F101.vap_outlet.temperature.unfix()
321 m.fs.H102.outlet.temperature.unfix()
322 m.fs.C101.condenser.condenser_pressure.unfix()
323 m.fs.C101.condenser.reflux_ratio.unfix()
324 m.fs.C101.reboiler.boilup_ratio.unfix()
325
326 # Set bounds on decision variable
327 m.fs.H101.outlet.temperature[0].setlb(500)
328 m.fs.H101.outlet.temperature[0].setub(600)
329 m.fs.R101.outlet.temperature[0].setlb(600)
330 m.fs.R101.outlet.temperature[0].setub(900)
331 m.fs.R101.volume[0].setlb(0)
332 m.fs.F101.vap_outlet.temperature[0].setlb(298)
333 m.fs.F101.vap_outlet.temperature[0].setub(450)
334 m.fs.H102.outlet.temperature[0].setlb(350)
335 m.fs.H102.outlet.temperature[0].setub(400)
336 m.fs.C101.condenser.condenser_pressure.setlb(101325)
337 m.fs.C101.condenser.condenser_pressure.setub(150000)
338 m.fs.C101.condenser.reflux_ratio.setlb(0.1)
339 m.fs.C101.condenser.reflux_ratio.setub(5)
340 m.fs.C101.reboiler.boilup_ratio.setlb(0.1)
341 m.fs.C101.reboiler.boilup_ratio.setub(5)
342
343 # Call solver again to optimize flowsheet
344 res = solver.solve(m, tee=True)
```

FIGURE 16 Unfixing variables, setting bounds and solving the optimization problem

- F101 outlet temperature [298, 450] K
- H102 outlet temperature [350, 400] K
- C101 condenser pressure [101 325, 150 000] Pa
- C101 reflux ratio [0.1, 5]
- C101 boilup ratio [0.1, 5]

4.10 | Step 10: Perform optimization and inspect solution

Thus, the flowsheet model is readily converted to an optimization problem. Since IPOPT was used to solve the flowsheet previously, the same solver can now be applied to solve the optimization problem as shown in line 344 of Figure 16. The optimization problem contains 1196 variables and 1189 constraints and solved in 42 iterations (0.91 s total, 0.18 s CPU time in IPOPT).Table 4 shows the stream table and results summary for the optimized process. As can be seen, the process met the increased product flowrate and purity demands of 0.18 mol/s at 99% purity. The optimization was also able to reduce the yearly operating cost by 4.5% from \$427 511/yr. to \$408 260/yr.; however, the amortized capital cost more than doubled from \$14 690/yr. to \$29 921/yr. due to the need to increase the conversion in the reactor (from 75% to 93%) to meet the increased production and purity constraints. The reactor volume required to achieve this conversion was 0.299 m^3.

Looking further into the results, it can be seen that some of the operating conditions have been pushed to their bounds. The product flow rate and purity are at the minimum values of 0.18 mol benzene/s and 99% which is expected as increasing recovery would require more energy (i.e. cost) to purify the product. The operating temperature of the flash separator (F101) has also been reduced to the lower bound in order to minimize the amount of benzene in the vapor stream, while the operating pressure for the distillation column is also at the lower bound, which will reduce the cooling costs required to achieve the target purity. The optimal reflux and boilup ratios in the distillation column (C101) are 0.801 and 1.135 respectively, which have changed from the initial values of 0.5.

4.11 | Step 11: Optimization under uncertainty

One of the key advantages of the IDAES-CMF is the ability to leverage the power of EO models along with the

TABLE 4 Stream table for the optimized process

Objective	$ 438 181/yr.				
Operating cost	$ 408 260/yr.				
Capital cost	$ 29 921/yr.				
Purity	99.00%				
Conversion	93.29%				
Stream	s01	s02	s03	s04	s05
Flow rate (mol/s)					
Liq, benzene	1e−8	1e−8	2.8E−08	4.5549E−08	4.5549E−08
Liq, toluene	0.3	1e−8	0.30000	7.5077E−07	7.5077E−07
Liq, hydrogen	1e−8	1e−8	2.8E−08	1.00E−08	1.00E−08
Liq, methane	1e−8	1e−8	2.8E−08	1.00E−08	1.00E−08
Vap, benzene	1e−8	1e−8	0.045541	0.045541	0.32631
Vap, toluene	1e−8	1e−8	9.5904e−4	0.30096	0.020190
Vap, hydrogen	1e−8	0.3	0.37693	0.37693	0.096158
Vap, methane	1e-8	0.02	1.2231	1.2231	1.5038
Temperature (K)	303.2	303.2	300.86	568.76	790.03
Pressure (Pa)	350 000	350 000	350 000	350 000	350 000
Stream	s06	s07	s08	s09	s10
Flow rate (mol/s)					
Liq, benzene	1.00E−08	1.00E−08	1.00E−08	1.00E−08	0.26938
Liq, toluene	1.00E−08	1.00E−08	1.00E−08	1.00E−08	0.018992
Liq, hydrogen	1.00E−08	1.00E−08	1.00E−08	1.00E−08	2.88E−07
Liq, methane	1.00E−08	1.00E−08	1.00E−08	1.00E−08	2.88E−07
Vap, benzene	0.056926	0.011385	0.045541	0.045541	1.00E−08
Vap, toluene	1.1988E−3	2.3975E−4	9.5902E−4	9.5902E−4	1.00E−08
Vap, hydrogen	0.096158	0.019232	0.076926	0.076926	1.00E−08
Vap, methane	1.5038	0.30077	1.2031	1.2031	1.00E−08
Temperature (K)	298.15	298.15	298.15	298.15	298.15
Pressure (Pa)	350 000	350 000	350 000	350 000	350 000
Stream	s11	s12	s13		
Flow rate (mol/s)	0.28838	0.18	0.10838		
Mole fraction					
Benzene	0.93414	0.99	0.84137		
Toluene	0.06586	0.01	0.15863		
Temperature (K)	368.79	353.52	356.67		
Pressure (Pa)	150 000	101 325	101 325		

many useful libraries available as part of the Python Programming Language to quickly extend the analysis and visualization capabilities. Due to the flexibility and open nature of the framework that allows full access to the underlying equations, the IDAES-CMF is well-suited to extend a deterministic optimization model to related, and more complex, tasks including parameter estimation, optimization under uncertainty, data reconciliation, and conceptual design. To demonstrate this advanced capability, the deterministic optimization of the HDA case study will be extended to incorporate uncertainty in the kinetic parameters (denoted as θ) by formulating a two-stage stochastic programming problem that minimizes the following objective:

$$Min\,[Capital\,Cost + E_\theta(Operating\,Cost)] \quad (13)$$

where $E_\theta()$ is the expected value over a probability distribution for the uncertain parameters θ. The purpose here is to demonstrate the ease of setting up a stochastic program (or multi-scenario problem) within IDAES-CMF in Python and is not an extensive demonstration of stochastic programming itself.

For this example, a normal distribution will be assumed on the reaction kinetic parameters $\theta \in \mathcal{N}(\mu,\sigma^2)$, with the Arrhenius constant ($\mu = 6.3 \times 10^{10}$, $\sigma = 9450 \times 10^5$) and the activation energy ($\mu = 217.6 \times 10^3, \sigma = 3264$). To set

```
# Import random fucntion from numpy
import numpy.random as rnd

# Set mean values for the uncertain parameters
arrhenius_mean = 6.3e+10
act_energy_mean = 217.6e3

# Set the number of scenarios to run
n = 100

# Create a Pyomo Concrete Model for the master problem
m = pyo.ConcreteModel()

# Add a master variable for the reactor volume
m.reactor_volume = pyo.Var()

# Create placeholder object for the objective expression
obj_expr = 0

# Create lists to hold realizations of the uncertain parameters
a_list = []
e_list = []

# Iteratively create scenarios
for i in range(n):
    # Build an instance of the flowsheet for each scenario
    mo = get_opt_model()

    # Attach instance to the master model
    setattr(m, 'scenario_{}'.format(i), mo)

    # Determine random realizations of uncertain parameters
    arrhenius = rnd.normal(
        arrhenius_mean, 0.015*arrhenius_mean)
    act_energy = rnd.normal(
        act_energy_mean, 0.015*act_energy_mean)

    # Append parameter values to lists
    a_list.append(arrhenius)
    e_list.append(act_energy)

    # Set parameter values in flowsheet instance
    mo.fs.reaction_params.arrhenius.fix(arrhenius)
    mo.fs.reaction_params.energy_activation.fix(act_energy)

    # Add a non-anticipativity constraint
    mo.non_antic = pyo.Constraint(
        expr=mo.fs.R101.volume[0] == m.reactor_volume)

    # Deactivate the objective function in flowsheet instance
    mo.fs.objective.deactivate()

    # Append scenario objective to overall objective expression
    obj_expr += 1/n * (mo.fs.capital_cost + mo.fs.operating_cost)

# Create objective function for master problem
m.obj = pyo.Objective(expr=obj_expr)

# Solve master problem
res = solver.solve(m, tee=True)
```

FIGURE 17 Example of running an optimization under uncertainty

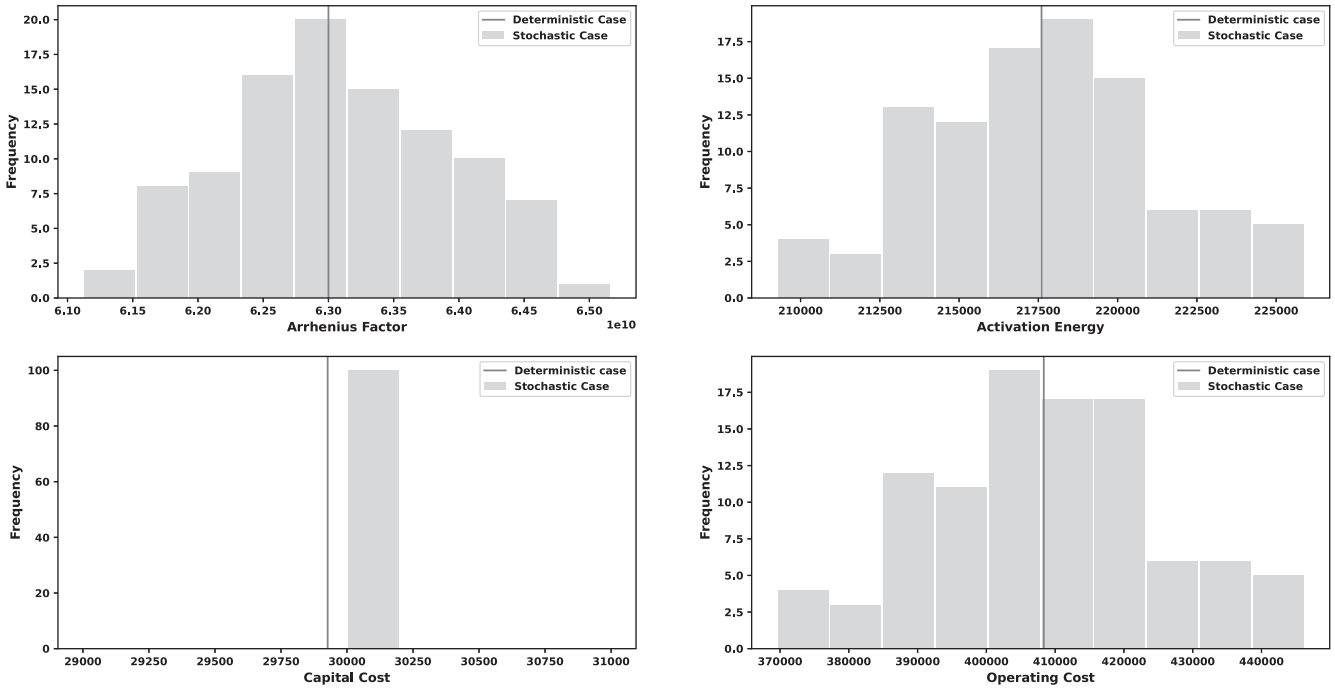

FIGURE 18 Distribution of capital and operating cost with respect to the uncertainty considered in the kinetic parameters

up this problem, the *rand* function from the *NumPy* Python package can be used to create the normally distributed parameter values and number of scenarios (100 in this example) as shown in lines 2 and 9, respectively, in Figure 17. To create the two-stage stochastic problem, a new *ConcreteModel* object is created, called "m," and then the (first-stage) design variable is declared, which is the reactor volume in this case. The second-stage decision variables, which differ in each scenario, are the outlet temperatures for H101, H102, F101 and R101 and the pressure, reflux and boilup ratios for C101. To set up the scenarios, the Python method *get_opt_model()* is used to return an instance of the deterministic optimization problem which was solved previously in step 10. This method is called for each of the 100 scenarios, and the model instance is attached to the *ConcreteModel* "m" that was created in line 12 in Figure 17. As each model instance is added, the *rand* function is used to assign values to the Arrhenius constant and the activation energy, as shown in lines 33 to 44 of Figure 17. A set of non-anticipativity constraints are then added to enforce the reactor volume to be the same across all scenarios (lines 47 and 48). The only remaining step setting up the problem is to deactivate the deterministic objective object (from lines 297–299 of Figure 15) and instead declare the objective expression for the stochastic problem as shown in lines 51–57 of Figure 17.

The resulting 2two-stage stochastic programming problem consists of 100 scenarios, 119 601 variables and 119 000 constraints and was solved with IPOPT in under 35 CPU seconds with an expected objective value of \$438 467/yr. By incorporating the uncertainty in the kinetic parameters, the amortized capital cost marginally increased from \$29 921/yr. to \$30 029/yr. Figures 18 and 19 show the distribution of the operating cost and the optimal values of the operating variables for the scenarios considered. From the solution of the stochastic program, it can be seen that reactor volume increased only slightly (to 0.300 m^3 from the deterministic solution volume of 0.299 m^3). For comparison, the reactor volume from the deterministic solution would have resulted in four infeasible scenarios, shown in Table 5, in which the overhead loss constraint (which limits the benzene to 20% in the vapor stream of the flash separator) was violated due to reduced single-pass conversion in the reactor. This simple example of optimization problem under uncertainty, demonstrates how the IDAES-CMF can easily extend deterministic optimization models to allow stochastic optimization under uncertainty—in this case with less than 20 lines of code.

While the case study shown here is relatively simple, the IDAES-CMF has been applied to simulating and optimizing complex chemical and energy related processes, such as the removal of CO_2 from power plant flue gas.[38] The IDAES-CMF has also been applied to a range of other problem types, such as conceptual design and process intensification[39] and dynamic real-time optimization and control,[40] and non-linear robust optimization.[41]

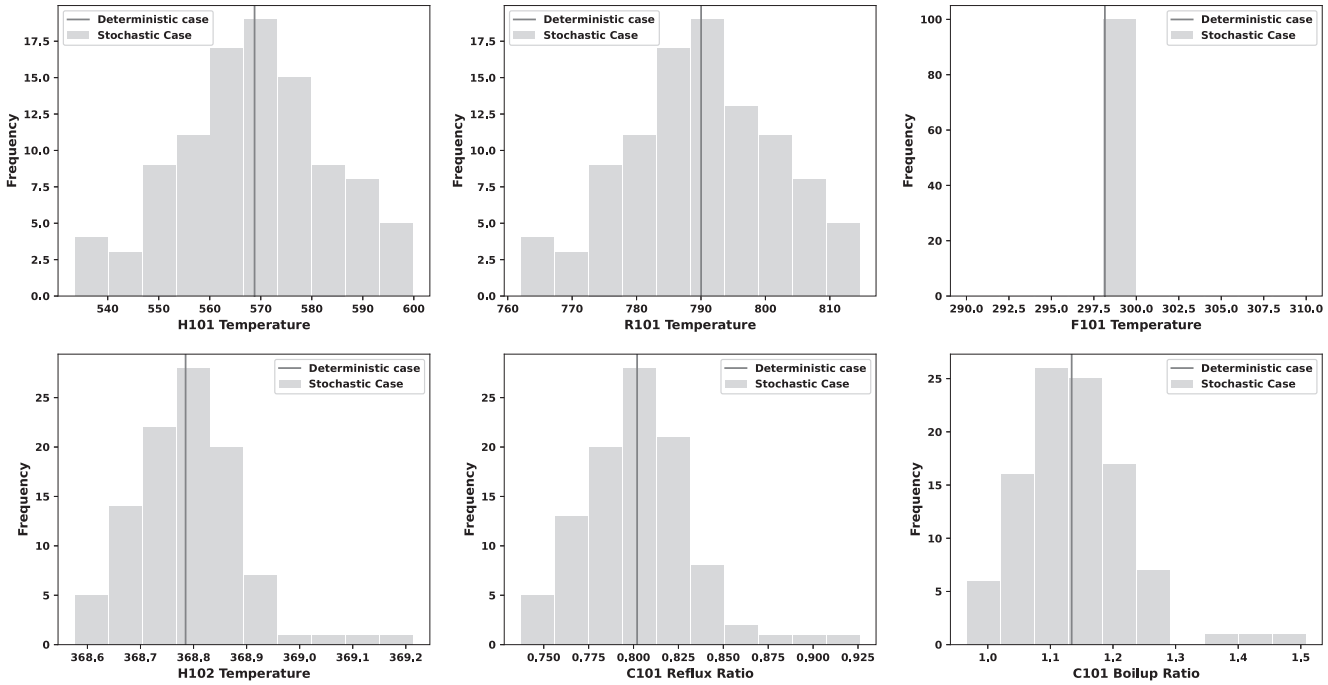

FIGURE 19 Distribution of the optimal values of the operating variables with respect to the uncertainty considered in the kinetic parameters

TABLE 5 Scenarios that were infeasible when using the deterministic solution from step 10

Scenario no.	Arrhenius factor (A)	Activation energy (E_a)
27	6.37×10^{10}	221.4×10^{3}
53	6.26×10^{10}	214.1×10^{3}
62	6.21×10^{10}	219.9×10^{3}
81	6.26×10^{10}	213.9×10^{3}

5 | CONCLUSION

The IDAES Core Modeling Framework integrates mathematical optimization and simulation capabilities with libraries of equation-based models of unit operations and thermophysical properties and tools for editing and creating new models. The IDAES-CMF builds on the foundation of Pyomo,[27] an open-source, extensible algebraic modeling environment implemented in the Python programming language which provides direct access to state-of-the-art solvers, solution techniques and, through Python, access to numerous libraries for scientific computing, data analyses and visualization capabilities. This is combined with fully modifiable and extensible libraries of common processing unit operations and property calculations which enable a seamless transition between dynamic and steady-state modeling. The IDAES-CMF allows tight integration with advanced optimization-based approaches, such as conceptual design and optimization under uncertainty, to support the full process modeling lifecycle from process design to dynamic optimization and control within a single computational platform. With these capabilities, IDAES-CMF can be utilized to design novel processes and technologies for large scale complex systems, with more complete optimization studies than ever before.

The IDAES-CMF has been demonstrated by applying it to model and optimize a flowsheet for the hydrodealkylation (HDA) of toluene to form benzene, which includes both chemical reactions and phase separation to separate benzene from toluene. The case study provides a simple demonstration of how the IDAES-CMF can be used to rapidly assemble an EO model of the process using a library of pre-built, reusable unit operation models and property packages. Furthermore, the case study demonstrates its capabilities for solving both deterministic and stochastic optimization problems. The HDA optimization example includes 7 degrees of freedom (total of 1173 variables and 1186 constraints), including the reflux and boilup ratios and operating pressure of the distillation column, demonstrating the ability of the framework to optimize a complex non-linear system. The case study also highlights the flexibility of the IDAES-CMF to include different levels of detail and rigor in different parts of the flowsheet by demonstrating how two different sets of property calculations and component lists can be included in the same

flowsheet by assuming complete removal of hydrogen and methane from the liquid product stream prior to the distillation column. Moreover, the IDAES-CMF framework allows these optimization models to be extended in a very flexible manner to consider a variety of challenging optimization formulations. For instance, it was demonstrated that with less than 20 lines of python code, the HDA example can be readily extended to incorporate uncertainty through a two-stage stochastic program with 117 601 variables and 117 000 constraints while remaining computationally tractable and solving in under 35 CPU s.

ACKNOWLEDGMENTS

The authors would like to recognize the contributions of the greater IDAES team to the development of the IDAES Integrated Platform, and the numerous contributors to the model libraries and associated tools. IDAES is supported through the Simulation-Based Engineering, Crosscutting Research Program within the U.S. Department of Energy's Office of Fossil Energy.

CONFLICT OF INTEREST

The authors declare that there are no conflicts of interest regarding the publication of this article.

AUTHOR CONTRIBUTIONS

Andrew Lee: Conceptualization; methodology; software; writing-original draft; writing-review & editing. **Jaffer Ghouse:** Conceptualization; methodology; software; writing-original draft; writing-review & editing. **John Eslick:** Conceptualization; methodology; software; writing-review & editing. **Carl Laird:** Conceptualization; methodology; software; supervision; writing-review & editing. **John Siirola:** Conceptualization; methodology; software; supervision; writing-review & editing. **Miguel Zamarripa:** Conceptualization; methodology; software; writing-original draft; writing-review & editing. **Dan Gunter:** Project administration; software; supervision; writing-original draft; writing-review & editing. **John Shinn:** Funding acquisition; project administration; supervision. **Alexander Dowling:** Conceptualization; methodology; project administration; supervision; writing-original draft; writing-review & editing. **Debangsu Bhattacharyya:** Conceptualization; methodology; project administration; supervision; writing-review & editing. **Lorenz Biegler:** Conceptualization; methodology; project administration; supervision; writing-original draft; writing-review & editing. **Anthony Burgard:** Conceptualization; funding acquisition; methodology; project administration; software; supervision; writing-original draft; writing-review & editing. **David Miller:** Conceptualization; funding acquisition; project administration; supervision; writing-original draft; writing-review & editing.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in the IDAES GitHub repository at https://github.com/IDAES/idaes-pse.

DISCLAIMER

This project was funded by the United States Department of Energy, National Energy Technology Laboratory, in part, through a site support contract. Neither the United States Government nor any agency thereof, nor any of their employees, nor the support contractor, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Contract No. DEAC02-05CH11231.

NOMENCLATURE

A	area
c_p	specific heat capacity
C	concentration
$\hat{E}$	volume-specific internal energy
F	flow rate
H	enthalpy flow
ΔH_{rxn}	molar heat of reaction
l	length
M	mass transfer
p	pressure
ΔP	pressure difference
Q	heat transfer
t	time
T	temperature
V	volume
W	work transfer
x	spatial domain
X_r	extent of reaction

GREEK LETTERS

α stoichiometric coefficient
γ flow direction multiplier
ν elemental composition
ρ density
ϕ volume fraction

SUBSCRIPTS

in inlet of control volume
out outlet of control volume
x point in spatial domain

INDEX

E elements in set E
i components in set I
p phases in set P
r reactions in set R

ORCID

Andrew Lee https://orcid.org/0000-0002-0631-2537
Jaffer H. Ghouse https://orcid.org/0000-0002-6352-6726
John C. Eslick https://orcid.org/0000-0002-7334-2277
Miguel A. Zamarripa https://orcid.org/0000-0001-8216-2203
Dan Gunter https://orcid.org/0000-0002-2779-2744
Alexander W. Dowling https://orcid.org/0000-0001-9383-7499
Lorenz T. Biegler https://orcid.org/0000-0003-3875-4441
David C. Miller https://orcid.org/0000-0002-7378-5625

ENDNOTE

[1] https://github.com/IDAES/examples-pse/tree/main/src/Tutorials/Basics

REFERENCES

[1] A. W. Dowling, V. M. Zavala, *Comput. Chem. Eng.* **2018**, *114*, 254. https://doi.org/10.1016/j.compchemeng.2017.09.018.
[2] DESIGN II. http://www.chemshare.com/.
[3] Aspen Hysys. https://www.aspentech.com/en/products/engineering/aspen-hysys.
[4] CHEMCAD. https://www.chemstations.com/.
[5] Aveva Group. Pro/II Process Engineering. **2019**.
[6] ProSim. ProSim.
[7] ASPENTech. ASPEN Engineering Suite.
[8] J. D. Parkins, R. W. H. Sargent, Speedup: A Computer Program for Steady-State and Dynamic Simulation and Design of Chemical Processes. in *AIChE Symposium Series*, AIChE, London **1982**, p. 1.
[9] P. C. Piela, T. G. Epperly, K. M. Westerberg, A. W. Westerberg, *Comput. Chem. Eng.* **1991**, *15*(1), 53. https://doi.org/10.1016/0098-1354(91)87006-U.
[10] R. Von Watzdorf, U. G. Naef, P. I. Barton, C. C. Pantelides, *Comput. Chem. Eng.* **1994**, *18*, S343. https://doi.org/10.1016/0098-1354(94)80057-X.
[11] C. C. Pantelides, P. I. Barton, *Comput. Chem. Eng.* **1993**, *17*, S263. https://doi.org/10.1016/0098-1354(93)80240-N.
[12] Process Systems Enterprise. gPROMS.
[13] P. Barton, C. Pantelides, *Simul. Ser.* **1993**, *25*, 25. https://yoric.mit.edu/gproms-combined-discretecontinuous-modelling-environment-chemical-processing-systems.
[14] J. Tolsma, P. I. Barton, *Ind. Eng. Chem. Res.* **2000**, *39*(6), 1826. https://doi.org/10.1021/ie990734o.
[15] J. E. Tolsma, P. I. Barton, *Comput. Aided Chem. Eng.* **2001**, *9* (C), 309. https://doi.org/10.1016/S1570-7946(01)80047-X.
[16] P. I. Barton, C. C. Pantelides, *AIChE J.* **1994**, *40*(6), 966. https://doi.org/10.1002/aic.690400608.
[17] S. Bublitz, E. Esche, G. Tolksdorf, V. Mehrmann, J. U. Repke, *Chem. Ing. Tech.* **2017**, *89*(11), 1503. https://doi.org/10.1002/cite.201700041.
[18] E. Esche, C. Hoffmann, M. Illner, D. Müller, S. Fillinger, G. Tolksdorf, H. Bonart, G. Wozny, J. U. Repke, *Chem. Ing. Tech.* **2017**, *89*(5), 620. https://doi.org/10.1002/cite.201600114.
[19] L. T. Biegler, *Curr. Opin. Chem. Eng.* **2018**, *21*, 32. https://doi.org/10.1016/j.coche.2018.02.008.
[20] S. Cremaschi, *Comput. Chem. Eng.* **2015**, *81*, 130. https://doi.org/10.1016/j.compchemeng.2015.05.007.
[21] J. Gong, F. You, *Curr. Opin. Chem. Eng.* **2015**, *10*, 77. https://doi.org/10.1016/j.coche.2015.09.001.
[22] F. Trespalacios, I. E. Grossmann, *Chem. Ing. Tech.* **2014**, *86*(7), 991. https://doi.org/10.1002/cite.201400037.
[23] T. A. N. Heirung, J. A. Paulson, J. O'Leary, A. Mesbah, *Comput. Chem. Eng.* **2018**, *114*, 158. https://doi.org/10.1016/j.compchemeng.2017.10.026.
[24] R. Fourer, D. M. Gay, B. W. Kernighan, *Manage. Sci.* **1990**, *36* (5), 519. https://doi.org/10.1287/mnsc.36.5.519.
[25] G.D. Corporation. General Algebraic Modeling Systerm (GAMS) Release 24.2.1. **2013**.
[26] Bisschop J. *AIMMS Optimization Modeling*. **2006**. http://download.aimms.com/aimms/download/manuals/AIMMS3_OM.pdf.
[27] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, JDS. in *Pyomo – Optimization Modeling in Python*, 2nd ed., Springer, Switzerland **2017**.
[28] J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, *SIAM Rev.* **2017**, *59*(1), 65. https://doi.org/10.1137/141000671.
[29] I. Dunning, J. Huchette, M. Lubin, *SIAM Rev.* **2017**, *59*(2), 295. https://doi.org/10.1137/15M1020575.
[30] A. Lee, J. H. Ghouse, Q. Chen, et al., A Flexible Framework and Model Library for Process Simulation, Optimization and Control. in *Computer Aided Chemical Engineering*, Vol. 44. Elsevier, Amsterdam **2018**, p. 937. https://doi.org/10.1016/B978-0-444-64241-7.50151-8.
[31] B. Nicholson, J. D. Siirola, J. P. Watson, V. M. Zavala, L. T. Biegler, *Math. Program. Comput.* **2018**, *10*(2), 187. https://doi.org/10.1007/s12532-017-0127-0.
[32] J. Douglas, *Conceptual Design of Chemical Processes*, McGraw-Hill, New York **1988**.
[33] A. P. Burgard, J. P. Eason, J. C. Eslick, et al., A Smooth, Square Flash Formulation for Equation-Oriented Flowsheet Optimization. in *Computer Aided Chemical Engineering*, Vol. 44. Elsevier, Amsterdam **2018**, p. 871. https://doi.org/10.1016/B978-0-444-64241-7.50140-3.
[34] R. C. Reid, J. M. Prausnitz, B. E. Poling, *The Properties of Gases and Liquids*, 4th ed., McGraw-Hill, New York **1987**.
[35] R. H. Perry, D. W. Green, J. O. Maloney, *Perry's Chemical Engineers' Handbook*, 7th ed., McGraw-Hill, New York **1997**.

[36] Pyomo Documentation. Accessed May 7, 2020. https://pyomo.readthedocs.io/en/stable/modeling_extensions/network.html.
[37] A. Wächter, L. T. Biegler, *Math. Program.* **2006**, *106*(1), 25. https://doi.org/10.1007/s10107-004-0559-y.
[38] P. Akula, J. Eslick, D. Bhattacharyya, D. C. Miller, *Ind. Eng. Chem. Res.* **2021**, *60*(14), 5176. https://doi.org/10.1021/acs.iecr.0c05035.
[39] E. Soraya Rawlings, Q. Chen, I. E. Grossmann, J. A. Caballero, *Comput. Chem. Eng.* **2019**, *125*, 31. https://doi.org/10.1016/j.compchemeng.2019.03.006.
[40] D. Thierry, L. T. Biegler, *AIChE J.* **2019**, *65*(7), e16511. https://doi.org/10.1002/aic.16511.
[41] N. M. Isenberg, P. Akula, J. C. Eslick, D. Bhattacharyya, D. C. Miller, C. E. Gounaris, *AIChE J.* **2021**, *67*, e17175. https://doi.org/10.1002/aic.17175.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: A. Lee, J. H. Ghouse, J. C. Eslick, C. D. Laird, J. D. Siirola, M. A. Zamarripa, D. Gunter, J. H. Shinn, A. W. Dowling, D. Bhattacharyya, L. T. Biegler, A. P. Burgard, D. C. Miller, *J Adv Manuf Process* **2021**, e10095. https://doi.org/10.1002/amp2.10095