Systems & Control Transactions

Research Article
Foundations of Computer Aided Process Design (FOCAPD 2024)
Breckenridge, Colorado, USA. July 14-18, 2024
Peer Reviewed Conference Proceeding

PSE PRESS

Model Diagnostics for Equation-Oriented Models: Roadblocks and the Path Forward

Andrew Lee[a,b]*, Robert B. Parker[c], Sarah Poon[d], Dan Gunter[d], Alexander W. Dowling[e], and Bethany Nicholson[f]

[a] National Energy Technology Laboratory, Pittsburgh, PA 15236, USA
[b] NETL Support Contractor, Pittsburgh, PA 15236, USA
[c] Los Alamos National Laboratory, Los Alamos, NM 87545, USA
[d] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
[e] University of Notre Dame, Notre Dame, IN 46556, USA
[f] Sandia National Laboratories, Albuquerque, NM 87185, USA
* Corresponding Author: andrew.lee@netl.doe.gov.

ABSTRACT

Equation-Oriented (EO) modeling techniques have been gaining popularity as an alternative for simulating and optimizing process systems due to their flexibility and ability to leverage state-of-the-art solvers inaccessible to many procedural modeling approaches. Despite these advantages, adopting EO modeling tools remains challenging due to the significant learning curve and effort required to build and solve models. Many techniques are available to help diagnose problems with EO process models and reduce the effort required to create and use them. However, these techniques still need to be integrated into EO modeling environments, and many modelers are unaware of sophisticated EO diagnostic tools. To survey the availability of model diagnostic tools and common workflows, the U.S. Department of Energy's Institute for the Design of Advanced Energy Systems (IDAES) has conducted user experience interviews of users of the IDAES Integrated Platform (IDAES-IP) for process modeling. The interviews reveal a gap between the availability and utilization of model diagnostic tools driven primarily by a lack of awareness of and lack of standard interfaces among different tools. To address this gap, the IDAES team has developed a recommended workflow for integrating diagnostics into the model development process and an IDAES Model Diagnostics Toolbox that provides a standard interface for many of these best practices. This paper identifies barriers to the widespread adoption of diagnostic tools for EO models and reduces these barriers by providing a standard, user-friendly interface for many different tools.

Keywords: Modelling and Simulations, Optimization, Simulation, Pyomo, Education

INTRODUCTION

Equation-Oriented (EO) modeling techniques are powerful tools for solving complex process engineering models, providing modelers with access to powerful gradient-based solvers and greater flexibility than more traditional sequential-modular tools [1]. Several EO-based modeling tools are now available for use, including ABACUS [2], AIMMS [3], AMPL [4], Aspen Custom Modeler [5], DAEPACK [6], GAMS [7], gPROMS [8], JuMP [9,10], and Pyomo [11].

However, EO-based tools come with several challenges, especially when applied to large models, as model tractability significantly depends on the formulation and the provision of good initial guesses to the solver. The number of potential issues in EO models is extensive, and understanding how to identify and remedy them during model development requires significant expertise from the user. Some possible problems include:

- inconsistencies in units of measurement,
- numerical evaluation errors and singularities,
- structural singularities in the problem matrix [12,13],
- degenerate equations and rank deficiency in the

Jacobian of the model [14],

- poorly scaled variables and constraints, and
- bound violations and infeasible constraints.

These issues are easily overlooked when manually inspecting large, complicated models such as those required for sophisticated process systems.

In 2015, the U.S. Department of Energy established the Institute for the Design of Advanced Energy Systems (IDAES) to develop a next-generation platform for process modeling: the IDAES Integrated Platform (IDAES-IP) [15,16]. IDAES aims to make state-of-the-art EO modeling techniques accessible to the process modeling community. A significant advantage of the IDAES-IP is that it is built on Pyomo [11] and Python, allowing the development of algorithms that directly interrogate a model's structural and numerical properties.

UNDERSTANDING CURRENT APPROACHES

To understand the challenges faced by modelers, the IDAES team started by performing a user experience study, interviewing a range of users of both IDAES-IP and Pyomo with varying backgrounds and levels of modeling experience. These interviews sought to discover each modeler's most common issues and the workflows and tools they used to help diagnose them. Additionally, the interviews sought to understand how modelers developed these workflows and learned about the different tools they applied to better inform the education of new modelers.

The interviews were structured as free-form conversations to capture the range and nuance of users' experiences. With encouragement, interviewees often expanded on their responses to an initial set of questions and provided relevant anecdotes. While much of each interview was open-ended, **Table 1** summarizes the responses to three interview questions. Note that the categorization of issues in responses to **Q1** is imperfect due to imprecise definitions of various modeling issues. For example, what one user may refer to as a "degeneracy" due to a flow rate approaching zero, another may refer to as a "scaling issue." In responses to **Q2** and **Q3**, "Basic 'model statistics'" refers to simple utility functions for identifying variables and constraints with useful properties, *e.g.*, variables close to (or exceeding) their bounds or constraints with large residuals, that are provided by the `idaes.core.util.model_statistics` module.

The most frequently identified issue was poor model scaling. This is partly because the IDAES-IP (and Pyomo) tools are designed to be as general as possible to cover a wide range of operating conditions. For best performance, modelers should specify scaling factors to tailor the problem to the specific operating range of interest.

Another common issue IDAES-IP users encounter is infeasibility, usually due to bound violations, partly due to the modular nature of the IDAES model libraries. Modelers frequently rely on pre-built sub-models, *e.g.*, for thermodynamics, but often do not check that the ranges of applicability, and thus bounds, for those sub-models match the needs of their application.

A third common issue was numerical evaluation errors, such as divisions by zero or logarithms of negative numbers. Nearly a quarter of the interviewees reported numerical evaluation errors related to zero-flow conditions in their models. These frequently arise in process systems involving reactions where not all components are present in every process stream. Suppose care is not taken to eliminate these components in streams where they do not exist or to ensure a minimum positive lower bound on flow/concentration. In that case, these values can cause numerical errors in complicated thermodynamic expressions involving logarithms or divisions.

Perhaps surprisingly, degenerate equations were not a common issue identified by users. However, this might indicate a need for more awareness of degeneracies rather than their infrequent occurrence. Most process flowsheets include at least one recycle stream, and these can easily introduce degeneracies into a model unless care is taken to identify an appropriate tear stream. Degeneracies are often tricky to identify unless the modeler knows what to look for. Degenerate models have a family of feasible solutions; thus, the solver frequently reports "optimal solution" termination messages. Hence, if the modeler does not intentionally inspect the solver logs, they may miss that their model is degenerate.

Qualitatively, the interviews revealed that most users had no formal training in diagnosing modeling issues and that most learning had come through experience. As a result, there was little uniformity in the workflows and tools used, and the interviewees tended to only think about issues they had previously encountered. Additionally, while most modelers interviewed used a bottom-up approach, starting with small models and gradually adding complexity and resolving issues as they appeared, some of those interviewed used a top-down approach, building the full complex model first and then trying to work backward to find the resulting issues.

A second finding was that there were two types of modelers within the IDAES user base: those who focused on developing models for unit operations and those who generally constructed flowsheets from pre-existing unit models in model libraries. Modelers in the first category were generally more aware of the different modeling issues they might encounter and how to fix them. At the same time, those who used existing models tended to struggle with diagnosing modeling issues. One key aspect of this is awareness of the model equations; unit

Table 1: Questions asked and responses received during 13 user experience interviews.

Q1. What are the issues you most commonly encounter with equation-oriented modeling and optimization in IDAES?

Response	Number of users
Scaling issues	8
Infeasibility	4
Incorrect degrees of freedom	4
Evaluation errors	4
Initialization issues	3
Zero-flow issues	3
Not understanding library models	3
Degeneracy	1
Incompatible assumptions	1

Q2. How do you recognize when these problems occur in your model or workflow?

Response	Number of users
Solver log	8
Basic “model statistics”	4
Inspect Jacobian	2
Incidence graph analysis	1
Test failure	1
Physical sanity check	1

Q3. What tools do you use to diagnose the cause of the issue or propose a fix?

Response	Number of users
Basic “model statistics”	8
Degeneracy Hunter	8
Inspect Jacobian	4
Change solver options	4
Solver log	3
Incidence graph analysis	2
Ask for help	2
Parameter sweep	2
Pyomo model viewer	2
Solver callback	1
Singular value decomposition	1
Serialize model state	1

model developers were generally fully aware of the equations in their models as they had written most equations themselves, while model users typically treated the library models as black boxes and were thus unable to resolve modeling issues that arose from them.

Finally, while solver logs were the most common method of identifying that a problem had occurred, many interviewees were not familiar with the full content of these. They focused mainly on apparent signs such as the final solver status, number of iterations, and obvious warning messages. This practice reveals a need to educate modelers about the data provided by solvers and how it can be used to identify potential modeling issues. Indeed, many modelers interviewed assumed that once a solver reports an “optimal solution,” that their model is well-behaved; it is crucial to educate modelers that just as one should always check that the solution of their model makes sense, they should also check the solver logs to ensure that no warnings were encountered.

In terms of tools used to help diagnose modeling issues, the modelers identified existing tools within the IDAES-IP toolset, Pyomo, or other Python packages (*e.g.*, the SciPy packages for linear algebra [17]) to diagnose most of the modeling issues encountered. These tools fell into three categories:

1. *Structural analysis* tools that analyze constraints, variables, and bounds without depending on specific variable values. These tend to be computationally cheap and do not require the model to be initialized.

2. *Numerical analysis* tools that analyze constraints, bounds, and their Jacobian at specific variable values. These tend to be slightly more computationally expensive and require that all variables have values from a previous solution or initialization.
3. *Advanced techniques*, such as singular value decomposition or meta-modeling techniques, to identify irreducible degenerate sets. These tend to be more computationally expensive.

However, as these tools were scattered across different packages with different interfaces and output formats, no single modeler who was interviewed was aware of all the tools and how to use them. When asked about other tools, the most common response was that modelers were unaware of them, followed by comments about the lack of user-friendliness of some tools.

The interviews reveal that the main barrier to effective debugging of EO models was awareness of different issues modelers might encounter, the tools available to diagnose them, and methods to remedy them when they occur. To address this, the IDAES team developed and documented a comprehensive workflow and set of best practices for diagnosing modeling issues, accompanied by an integrated toolbox interfacing with most of the existing tools mentioned by the interviewees.

PROPOSED DIAGNOSTICS WORKFLOW

The interviews revealed that issues can arise at many points during the modeling process. Therefore, modelers must continuously iterate between model development and diagnostic testing. Almost all the modelers interviewed agreed that the best approach to model development was to start from the bottom up, starting from simpler sub-models and gradually adding complexity, checking for and resolving issues at each step. In this way, the source of the problems can be easily traced to the most recent changes made to the model, and the smaller these changes, the easier it is to identify the root cause. As such, model development and diagnosis should always be viewed as an incremental process of gradual development and refinement, not a "one-shot" fix-all operation.

Additionally, the consensus across the modelers interviewed was that diagnosis should start with a "square" simulation model with no degrees of freedom; all optimization models derive from an underlying deterministic model by releasing some degrees of freedom. Adding degrees of freedom to the problem can often make it easier to solve, but this can obscure issues in the model. If the simulation model is ill-posed or poorly formulated, then these issues remain in the optimization model even if they are not apparent.

Based on this incremental, iterative approach and the "hierarchy" of diagnostic tools, a sequence of common steps for model development was proposed, which is shown in **Figure 1** and explained below.

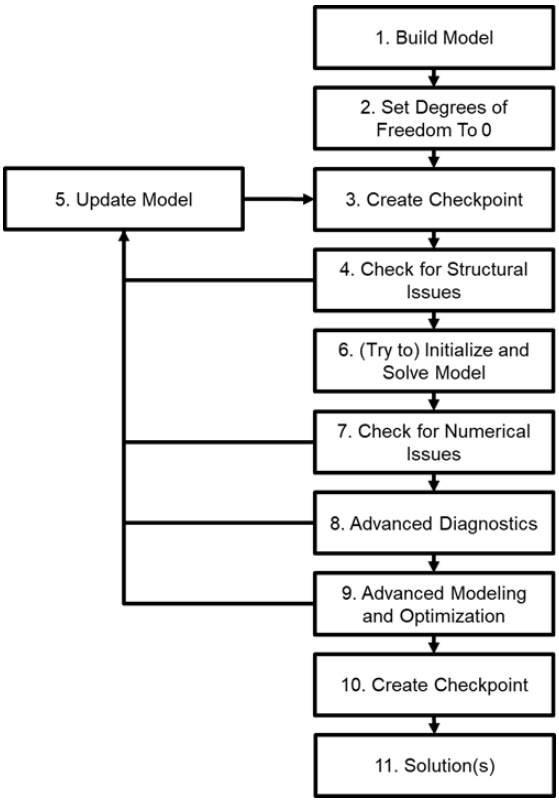

Figure 1. Proposed Model Diagnostics Workflow.

1. The workflow begins with an existing model, which should be the simplest possible model to start with (or an existing model that has been thoroughly tested for issues).
2. Next, the degrees of freedom necessary to form a square simulation model should be fixed. If the modeler is uncertain of which degrees of freedom to choose, they should take their best guess and allow the subsequent checks to identify any potential singularities.
3. As the model development and diagnosis process is iterative and often involves trial and error, modelers should always take the time to create a checkpoint so that they can easily revert changes to earlier states in case the current thrust of development becomes a dead-end. This is especially important results are being generated and recorded in parallel with the model development; there is nothing more frustrating than to discover that you can no longer reproduce a previous set of results with no idea of how or why this occurred. (Tip: version control systems such as git make it much easier to systematically track incremental model changes during

iterative diagnostics and refinement.)

4. When checking for potential modeling issues, the modeler should first apply structural analysis tools. As these only rely on model structure, they do not require an initial guess for the model solution. Fixing these structural issues before calling a solver decreases the chance of a critical solver failure.
5. Modelers should resolve issues as they are identified and should generally only make changes to address one issue at a time. Issues can often be interwoven, and fixing one issue may create or reveal other issues (or resolve other known issues). As each change is made to the model, modelers should return to the top of the workflow and follow all the steps again to make sure they are not accidentally introducing new issues.
6. Only once all (known) structural issues are resolved should modelers attempt to solve their model (initializing as necessary). Modelers should always review the solver output logs, as these can contain helpful information for diagnosing modeling issues. Hopefully, the solver will return a solution, even if it is infeasible, which can be used to continue the model diagnostics workflow. (Tip: If a solution is not returned, then the modeler can re-run the solver with a lower iteration limit or use a block triangular decomposition solve (see Section 3.6 of [12]) to obtain a partial solution.)
7. Once an initial solution is obtained, even if it is infeasible, the modeler can check for potential numerical issues at the current state. Again, problems should be resolved as they are identified, and modelers should return to the top of the workflow after each change. Numerical issues inherently depend on the current state of the model. As optimization involves exploring the behavior of the model over an entire design space, it is essential to perform numerical checks across this design space.
8. If issues remain (*e.g.*, failure to converge or signs of poor behavior in the solver logs), modelers should then apply the advanced diagnostics tools to try to find the root cause of the issues. As these techniques are computationally intensive, they are reserved for the final step to address challenging issues.
9. Once a reliable model has been identified, modelers can then move on to optimization studies and other more complex modeling tasks. Modelers should continue monitoring the solver logs for signs of potential issues and should apply the model diagnostics workflow if these signs appear.
10. Finally, as solutions and results are generated, modelers should always maintain a checkpoint and log of the model state (e.g., git commits) to ensure that they can reproduce these results in the future and to provide a point of comparison in case future changes to the model change these results.

Finally, modelers should keep in mind that their model may be used by others in the future who may not have the same level of familiarity with the model equations and formulation; this is especially important for developers of model libraries such as those in IDAES-IP. As the original modeler has the most experience with the formulation of the model, they are best suited for ensuring the model is as robust as possible to minimize the challenges faced by future users.

IDAES DIAGNOSTICS TOOLBOX

Whilst the workflow described above provides modelers with a series of steps to take when developing their models and diagnosing any issues that arise, they also need access to a suite of tools to perform these checks. To limit the number of methods and interfaces modelers need to know, a new Diagnostics Toolbox has been added to the IDAES-IP (v2.2.0, released September 2023) [16], which incorporates tools to check for a wide range of common modeling issues.

To assist users with the model diagnostics workflow, the Diagnostics Toolbox guides them through the diagnostics process through a series of "reporting methods." These reporting methods call a series of tools to check for different types of issues and then provide the modeler with a summary of the issues found and a set of next steps for further investigation. Each of these "next steps" corresponds to a "display" method to call one of the underlying methods and provides a detailed description of the issue and, if possible, its source.

Currently, the Diagnostics Toolbox contains three reporting methods—one for structural issues, one for numerical issues, and one for irreducible degenerate sets—and approximately 22 associated display methods. To assist modelers with identifying the most likely causes of model failures, problems are further divided into "warnings" representing issues that will most likely result in solver issues or incorrect results (*e.g.*, singularities and degeneracies), and "cautions" that warrant checking but may be expected model behavior (*e.g.*, variables which do not appear in any constraint). The Diagnostics Toolbox may be imported from IDAES as:

```
from idaes.core.util import DiagnosticsToolbox
```

The remainder of this section gives a brief overview of the structural and numerical issues that can be diagnosed by the Diagnostics Toolbox.

Structural Issues

The Diagnostics Toolbox contains three methods that check a model for unexpected variable status:

- variables which do not appear in any constraints,
- variables which are not declared as part of the model being diagnosed,
- and variables with a value fixed to exactly 0.

If any such variables are encountered, the modeler is cautioned. Additionally, the Diagnostics Toolbox contains methods to check constraints and objectives for consistent units, verify that the over and under-constrained subsystems of the model's Dulmage-Mendelsohn decomposition [12,13] are empty, and check for potential evaluation errors. If any of these checks fail, a warning is displayed.

Numerical Issues

The Diagnostics Toolbox contains several checks performed with simple variable and expression evaluations. These check for:

- variables with values at or outside their declared bounds,
- variables with no values,
- variables with values near zero,
- variables with extremely large or small values,
- variables with values approaching their bounds,
- and constraints with large residuals.

These checks display a warning or a caution depending on the magnitude of *e.g.*, variable value or bound proximity/violation. A variable with no value always yields a caution. The Diagnostics Toolbox also contains methods that analyze the model's Jacobian matrix to detect the following indicators of ill-conditioning:

- values in the Jacobian matrix with extremely large or small values,
- rows and columns in the Jacobian matrix with extreme 2-norm values,
- and condition number of the Jacobian.

Whether a warning or a caution is displayed again depends on the magnitude of the value detected.

Advanced Diagnostics Tools

For cases where the above tools are not sufficient to fully diagnose the issues in a model, the Diagnostics Toolbox also contains an implementation of Degeneracy Hunter [14] and an interface for performing Singular Value Decomposition (SVD) analysis on the model (by default using the SciPy linear algebra libraries [17], but with support for modeler-provided SVD routines).

CASE STUDY

To better demonstrate the application and utility of the Diagnostics Toolbox, consider the following toy problem with variables $\mathrm{v}_1[\text{units: } m]$, $\mathrm{v}_2[m]$, $\mathrm{v}_3 \in [0,5]$, v_4, $\mathrm{v}_5 \in [0,1]$, v_6, $\mathrm{v}_7 \in [0m, 1m]$, and v_8, and constraints:

$$\mathrm{v}_1 + \mathrm{v}_2 = 10 \quad (1)$$

$$v_3 = v_4 + v_5 \quad (2)$$

$$2v_3 = 3v_4 + 4v_5 + v_6 \quad (3)$$

$$\mathrm{v}_7 = 1 \times 10^{-8}\mathrm{v}_1 \quad (4)$$

One common issue facing modelers relying on prebuilt EO models is identifying suitable degrees of freedom to fix for their application. Poor choices can often result in infeasible models or structural singularities. To demonstrate this, consider setting the following conditions for fixed variables: $v_4 = 2$, $v_5 = 2$, and $v_6 = 0$.

While the above problem may appear relatively simple and involve only linear constraints, the model will yield an infeasible result after fixing the specified degrees of freedom and attempting to solve it. This is because there are several issues in the model as posed. As this model is small, an experienced modeler can likely identify most or all these issues by manually inspecting the variables and constraints; however, it will likely take a few minutes. The Diagnostics Toolbox, however, can automatically detect all these issues in under a second and direct the modeler to the root causes.

Following the diagnostics workflow, the first step is to check the model for structural issues. This is done by calling the `report_structural_issues` method from the Diagnostics Toolbox. This executes the series of checks described in the previous section, which reveals two warnings (critical issues) and two cautions.

Structural warning 1: Checking unit consistency reveals an issue in constraint 1): v_1 and v_2 were assigned units of meters; however, the constant in the constraint was not assigned units. If this constant was in units other than meters, then the solution to the model would be incorrect. This should be corrected by rewriting constraint 1) to include the units for the constant and applying any necessary unit conversion.

Structural warning 2: Performing a Dulmage-Mendelsohn partition of the incidence matrix reveals a structural singularity. After fixing v_4, v_5, and v_6, constraints 2) and 3) include only a single free variable. Similarly, constraints 1) and 4) include three free variables, and thus this sub-system is under-constrained. Addressing this issue requires the modeler to rethink the degrees of freedom they have chosen or the constraints they have chosen to write. The Dulmage-Mendelsohn partition helps the modeler choose degrees of freedom, as variables that are candidates for degrees of freedom must be in the under-constrained subsystem.

Structural caution 1: A simple inspection routine reveals that v_8 does not appear in any constraint. This could be considered a modeling error. Either it was meant to be included in a constraint, but a mistake was made, or this variable is extraneous and should be removed from the model definition.

Structural caution 2: v_6 has been assigned a value of exactly 0. While this will not cause an issue in the current model, modelers should be careful when assigning values of zero to ensure this will not cause numerical issues (*e.g.*, appearing in a denominator or logarithm).

For this example, assume that the units of the constant in constraint 1) were meters and that the modeler decides to use $v_2 = 5$ instead of $v_4 = 2$. After reviewing the two cautions, v_8 should be removed from the model, while v_6 can remain fixed to 0 for now. After making these changes and confirming that the structural issues have been resolved, the next step is to try to solve the model. The solver reports that the model is infeasible, so the next step is to check the model for numerical issues by calling the `report_numerical_issues` method from the Diagnostics Toolbox. This will execute the second set of checks described above, which will reveal two additional warnings and four cautions.

Numerical warning 1: At least one of the constraints has a large residual. The specific constraint may depend on the solver used but will be constraints 2) or 3). This should not be unexpected due to the infeasible solution returned by the solver.

Numerical warning 2: Iterating over all the variables in the model and comparing their values to any bounds specified will reveal that v_3 and v_5 have values at or outside their bounds. In the case of v_3, this is because the value required to satisfy both constraints 2) and 3) lies below the lower bound of 0, while for v_5 the issue is that it was fixed to a value outside of its bounds (it was assigned a value of 5 but had bounds of [0,1]). Addressing these issues requires the modeler to consider why these bounds were placed on the model in the first place (*e.g.*, to indicate a range over which an empirical correlation is reliable or to protect against a numerical singularity) and to decide whether the bounds can be relaxed or if changes to the model constraints are required.

Numerical cautions 1-4: All four cautions arise because $v_7 = 5 \times 10^{-8}$, which is likely close to the solver tolerance. Additionally, while it does not violate the bounds, this value is very close to the lower bound. Analysis of the Jacobian will reveal a large value for v_1 in constraint 4). This points to poor scaling of v_7 and constraint 4) and addressing this will likely improve solver performance.

As mentioned previously, an experienced modeler would likely discover most of these issues by manual inspection. However, this becomes more challenging for more complex (and especially nonlinear) models or modelers with less experience. The Diagnostics Toolbox runs these checks automatically and quickly, providing the modeler with immediate guidance on which constraints and variables to investigate to resolve any issues observed with solver behavior or solutions.

A full demonstration of applying the Diagnostics Toolbox to this example can be found here:

https://idaes-examples.readthedocs.io/en/latest/docs/diagnostics/diagnostics_toolbox_doc.html

ONGOING WORK

Since the release of the IDAES Diagnostics Toolbox, the team has been working to advertise these capabilities to users and solicit feedback on how they can be improved. The IDAES team have begun using these tools to debug modeling issues, and future work will involve a second round of user interviews to determine whether the tools have been successful. The team is also continuing to identify additional modeling issues and tools to diagnose them, as well as visualization tools to improve user-friendliness and aid in interpreting results.

Anecdotally, initial responses have been uniformly positive, and including a handful users who have successfully applied the tools in practice during the past few months. Users have reported that the Diagnostics Toolbox has greatly reduced the time required to get their models to solve. One user reported that the toolbox helped them resolve a structural singularity due to misspecified initial and boundary conditions in a discretized PDE model that they described as "like searching for a needle in a haystack."

However, diagnosing issues with nonlinear optimization problems remains challenging with many open questions. To quote an experienced researcher in process systems engineering consulted for this study:

> *Automatically diagnosing arbitrary issues with nonlinear optimization problems is a Holy Grail of Operations Research.*

While identifying the presence of an issue in a model can be automated, tracing this back to the true root cause and identifying a fix often requires a degree of engineering knowledge that can only be gained by experience. Documentation to help users in this process is still required, and a long-term goal for IDAES is to create a library of examples for resolving different issues.

CONCLUSIONS

The IDAES Diagnostics Toolbox and workflow have been developed to assist modelers with addressing issues encountered when developing, debugging, and solving equation-oriented models. A standard workflow for systematically identifying issues has been created.

This workflow is supported by a toolbox that includes methods for automatically detecting common modeling issues and identifying the associated variables and constraints for the modeler. This workflow and toolbox will significantly reduce the time, effort, and expertise that modelers require for diagnosing issues when working with large, complicated models.

ACKNOWLEDGEMENTS AND DISCLAIMER

This work was conducted as part of the U.S. Department of Energy's Institute for the Design of Advanced Energy Systems (IDAES), supported by the Office of Fossil Energy and Carbon Management's Simulation-based Engineering/Crosscutting Research Program.

This work was supported by the U.S. Department of Energy through the Los Alamos National Laboratory (LANL) LDRD program and the Center for Nonlinear Studies (CNLS). Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy (Contract No. 89233218CNA000001).

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-NA-0003525.

This project was funded by the Department of Energy, National Energy Technology Laboratory an agency of the United States Government, through a support contract. Neither the United States Government nor any agency thereof, nor any of its 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.

REFERENCES

1. Dowling A. W., Biegler L. T., A Framework for Efficient Large Scale Equation-Oriented Flowsheet Optimization. *Comput. Chem. Eng.* 72:3-20 (2015) https://doi.org/10.1016/j.compchemeng.2014.05.013
2. Tolsma J. E., Barton P. I., Process Simulation and Analysis with Heterogeneous Models. *Comput Aided Chem Eng.* 9(C):309-314 (2001) doi:10.1016/S1570-7946(01)80047-X
3. Bisschop J., AIMMS Optimization Modeling (2006) http://download.aimms.com/aimms/download/manuals/AIMMS3_OM.pdf
4. Fourer R., Gay D. M., Kernighan B. W., A Modeling Language for Mathematical Programming. *Manage Sci.* 36(5):519-554 (1990) doi:10.1287/mnsc.36.5.519
5. ASPENTech. ASPEN Engineering Suite
6. Tolsma J., Barton P. I., DAEPACK: An Open Modeling Environment for Legacy Models. *Ind Eng Chem Res.* 39(6):1826-1839 (2000) doi:10.1021/ie990734o
7. G.D. Corporation, General Algebraic Modeling System (GAMS) Published online (2013)
8. Barton P., Pantelides C., gPROMS - a Combined Discrete/Continuous Modelling Environment for Chemical Processing Systems. Simul Ser. 25:25-34. (1993) https://yoric.mit.edu/gproms-combined-discretecontinuous-modelling-environment-chemical-processing-systems
9. Bezanson J., Edelman A., Karpinski S., Shah V. B., Julia: A Fresh Approach to Numerical Computing. *SIAM Rev.* 59(1):65-98 (2017) doi:10.1137/141000671
10. Dunning I., Huchette J., Lubin M., JuMP: A Modeling Language for Mathematical Optimization. *SIAM Rev.* 59(2):295-320 (2017) doi:10.1137/15M1020575
11. Bynum M. L., Hackebeil G.A., Hart W. E., Laird C. D., Nicholson B. L., Siirola, J. D., Watson J.-P., Woodruff D. L., Pyomo – Optimization Modeling in Python. 3rd Edition. Springer (2021)
12. Parker R. B., Nicholson B. L., Siirola J. D., Biegler L. T., Applications of the Dulmage–Mendelsohn Decomposition for Debugging Nonlinear Optimization Problems, *Comput. Chem. Eng.* 178:108383 (2023) https://doi.org/10.1016/j.compchemeng.2023.108383
13. Dulmage, A. L., Mendelsohn, N. S., Coverings of bipartite graphs. *Can. J. Math.* 10:517–534. (1958) doi:10.4153/cjm-1958-052-0
14. Dowling A. W., Biegler L. T., Degeneracy Hunter: An Algorithm for Determining Irreducible Sets of Degenerate Constraints in Mathematical Programs. *Comput. Aided Chem. Eng.* 37:809 –814 (2015)
15. Lee A., Ghouse J. H., Eslick J. C., Laird, C. D., Siirola J. D., Zamarripa M. A., Gunter D., Shinn J. H., Dowling, A. W., Bhattacharyya D., Biegler L. T., Burgard A. P., Miller D. C., The IDAES Process Modeling Framework and Model Library—Flexibility

for Process Simulation and Optimization. *J. Adv. Manuf. Process.* 3(3):e10095 (2021) https://doi.org/10.1002/amp2.10095
16. IDAES. https://github.com/IDAES/idaes-pse
17. Virtanen P., Gommers R., Oliphant T. E., Haberland M., Reddy T., Cournapeau D., Burovski E., Peterson P., Weckesser W., Bright J., van der Walt S. J., Brett M., Wilson J., Millman K. J., Mayorov N., Nelson A. R. J., Jones E., Kern R., Larson E., Carey C. J., Polat I., Feng Y., Moore E. W., VanderPlas J., Laxalde D., Perktold J., Cimrman R., Henriksen I., Quintero E. A., Harris C. R., Archibald A. M., Ribeiro A. H., Pedregosa F., van Mulbregt P., and SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17(3), 261-272 (2020)

© 2024 by the authors. Licensed to PSEcommunity.org and PSE Press. This is an open access article under the creative commons CC-BY-SA licensing terms. Credit must be given to creator and adaptations must be shared under the same terms. See https://creativecommons.org/licenses/by-sa/4.0/