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

Jacobian-based Model Diagnostics and Application to Equation Oriented Modeling of a Carbon Capture System

Douglas A. Allan[ab]*, Anca Ostace[ab], Andrew Lee[ab], Brandon Paul[ab], Anuja Deshpande[ab], Miguel A. Zamarripa[ab], Joshua C. Morgan[ab] and Benjamin P. Omell[a]

[a] National Energy Technology Laboratory, Pittsburgh, PA 15236, USA
[b] NETL Support Contractor, Pittsburgh, PA 15236, USA
*Corresponding author: douglas.allan@netl.doe.gov

ABSTRACT

Equation-oriented (EO) modeling has the potential to enable the effective design and optimization of the operation of advanced energy systems. However, advanced modeling of energy systems results in a large number of variables and non-linear equations, and it can be difficult to search through these to identify the culprit(s) responsible for convergence issues. The Institute for the Design of Advanced Energy Systems Integrated Platform (IDAES-IP) contains a tool to identify poorly scaled constraints and variables by searching for rows and columns of the Jacobian matrix with small L^2-norms so they can be rescaled. A further singular value decomposition can be performed to identify degenerate sets of equations and remaining scaling issues. This work presents an EO model of a flowsheet developed for post-combustion carbon capture using a monoethanolamine (MEA) solvent system as a case study. The IDAES diagnostics tools were successfully applied to this flowsheet to identify problems to improve model robustness and enable the optimization of process design and operating conditions of a carbon capture system.

Keywords: Pyomo, Optimization, Carbon Dioxide Capture, Jacobian, Modelling

INTRODUCTION

In order to achieve carbon neutrality by 2050, as is the US Department of Energy's present goal [1], a wide variety of energy systems must be deployed. Process optimization can help allocate resources in the most efficient manner to effect the changes necessary in the US and world economies to achieve this goal. The Institute for the Design of Advanced Energy Systems (IDAES) was founded in 2015 to study such advanced energy systems and to develop the IDAES Integrated Platform (IDAES-IP) to facilitate their development and optimization. [2] IDAES-IP is based on the Pyomo modeling language [3-4] in Python and has been used to simulate a wide variety of chemical and energy process systems.

Because IDAES-IP is equation-oriented (EO), it can provide improved convergence when closing recycle loops over the sequential-modular (SM) approach that is used in popular commercial process modeling and simulation tools like Aspen Plus®. [5] However, it also requires more user skill to ensure that the model is well-formulated to benefit from these theoretical convergence improvements. IDAES has developed and implemented a diagnostics toolbox leveraging several years of experience of many experts in EO modeling, debugging, and optimization, making it available to the public. [6] In this work, we detail some of the techniques that have been incorporated in that toolbox as applied to a flowsheet being developed for analysis of post-combustion carbon capture (PCC) systems.

METHODS

When prototyping a model, it is typically best to solve a "square problem," i.e., one in which there are an equal number of free variables and equality constraints. The number of degrees of freedom can be checked by the function `idaes.core.util.model_statistics.degress_of_freedom`. There are zero degrees of freedom in a square problem. Variable bounds can be present, as they are often helpful to keep the nonlinear solver from exploring areas with non-physical solutions

or areas in which equations become undefined. For example, if a logarithm of a variable is taken, its lower bound should be set to zero.

However, variable bounds *should not* be active at a solution, because the inclusion of such a bound is effectively a constraint, reducing the number of degrees of freedom by 1. The system of equations then either becomes infeasible, which precludes a solution, or degenerate, which means that some of the equations are redundant. Different nonlinear solvers handle degeneracy differently, but it either prevents or dramatically slows convergence to a solution. For this reason, inequality constraints more complex than variable bounds should be avoided when formulating a square problem.

The principal diagnostic methods used in this work utilize the Jacobian. When solving a root-finding problem

$$\begin{bmatrix} f_1(x_1, \dots, x_n) \\ \vdots \\ f_n(x_1, \dots, x_n) \end{bmatrix} \coloneqq \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0} \tag{1}$$

the Jacobian matrix of $\boldsymbol{f}(\boldsymbol{x})$ is given by

$$J(\boldsymbol{x}) \coloneqq \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{2}$$

For a square problem, the Jacobian matrix is square. Each row of $J(\boldsymbol{x})$ corresponds to an equality constraint and each column corresponds to a free variable. Therefore, we can examine $J(\boldsymbol{x})$ to find clues about problems in equations and constraints. In the past, calculation of the Jacobian was difficult because derivatives had to either be calculated by hand or by finite differences. However, advances in algorithmic differentiation (AD) allow for automatic, precise calculation of derivatives. Pyomo offers access to the AD capacities of the AMPL solver library (ASL) [7] through the PyNumero interface [8].

The most common methods of solving multivariate root-finding problems of the form of (1), like the one utilized in IPOPT [9], which is freely available and was used as a nonlinear solver in this paper, are variations on Newton's Method. The method approximates $\boldsymbol{f}(\cdot)$ as linear and then repeatedly solves the linearized equation

$$\boldsymbol{f}(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k = 0 \tag{3}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k \tag{4}$$

until the condition

$$\|\boldsymbol{f}(\boldsymbol{x}_{k+1})\|_2 \leq \varepsilon \tag{5}$$

in which $\|\cdot\|_2$ is the vector two (Euclidean) norm, is satisfied for some chosen tolerance ε. In a practical implementation, the Newton step (4) may be truncated, either to account for variable bounds or because $\|\boldsymbol{f}(\boldsymbol{x}_{k+1})\|_2 > \|\boldsymbol{f}(\boldsymbol{x}_k)\|_2$, but it is desirable to take the full step when possible. It is a sign of a well-formulated method working on a well-formulated problem when the full step (4) is frequently taken until a solution is reached.

A condition number of a matrix is a measure of the sensitivity of changes in the solution to a system of linear equations. For example, for the Newton step-finding problem

$$J(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k = -\boldsymbol{f}(\boldsymbol{x}_k) \tag{6}$$

suppose we perturb the function output by $\delta\boldsymbol{f}_k$ and want to estimate the perturbation in the resulting step $\delta\boldsymbol{x}_k$

$$J(\boldsymbol{x}_k)(\Delta\boldsymbol{x}_k + \delta\boldsymbol{x}_k) = -(\boldsymbol{f}(\boldsymbol{x}_k) + \delta\boldsymbol{f}_k) \tag{7}$$

Use of the condition number for the 2-norm $\kappa_2(J(\boldsymbol{x}_k))$ gives us the bound

$$\|\delta\boldsymbol{x}_k\|_2 \leq \kappa_2\big(J(\boldsymbol{x}_k)\big)\|\delta\boldsymbol{f}_k\|_2 \tag{8}$$

In practice, such perturbations in the function output always exist from the roundoff errors in floating point arithmetic. The relative error inherent in double precision floating point arithmetic is on the order of 10^{-16}. Additionally, a large condition number $\kappa_2(J(\boldsymbol{x}_k))$ (greater than about 10^8) makes it difficult to solve the problem (6) numerically, increasing the time that Newton iterations take to solve.

The condition number of $J(\boldsymbol{x})$ is given by

$$\kappa_2\big(J(\boldsymbol{x})\big) = \|J(\boldsymbol{x})\|_2 / \|J(\boldsymbol{x})^{-1}\|_2 \tag{9}$$

in which $\|\cdot\|_2$ is the operator norm induced by the vector 2-norm, provided $J(\boldsymbol{x})$ is full rank. A more useful formula comes from the singular value decomposition (SVD). We factorize

$$J(\boldsymbol{x}) = U\Sigma V^T \tag{10}$$

in which U and V are orthogonal matrices and

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n \end{bmatrix} \tag{11}$$

in which $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. If $J(\boldsymbol{x})$ is full rank, then $\sigma_n > 0$ and

$$\kappa_2\big(J(\boldsymbol{x})\big) = \sigma_1/\sigma_n \tag{12}$$

Therefore, we can improve the problem formulation by reducing the condition number by bringing the values of σ_1 and σ_n closer together. This can be done by appropriate scaling of variables and constraints.

Scaling

When working in SI units, it is common to have variables and equations that vary over many orders of magnitude, from mole fractions of trace components with magnitudes 10^{-6} to enthalpy flow values that can be up to 10^6. IDAES-IP offers a variety of tools to assist in scaling both variables and constraints. Scaling serves two purposes: first, to ensure that the convergence criterion

(5) guarantees that all equations are satisfied without being unduly difficult to satisfy, and second, to reduce the condition number of the Jacobian. A method exists [10] for calculating scaling factors for variables and constraints that minimize the condition number of the Jacobian by solution of a convex program. While this method achieves the second goal, it has no guarantee of achieving the first goal.

Let there be two example equations that may occur in, for example, a heater:

$$y_{trace,in} - y_{trace,out} = f_1(\boldsymbol{x}) \tag{13}$$

$$H_{in} + Q - H_{out} = f_2(\boldsymbol{x}) \tag{14}$$

in which y_{trace} is the mole fraction of a trace component, H represents stream enthalpies, and Q is a heat duty. If the initial values were off by a relative error of 100%, the error in (13) would be of the order 10^{-6} and the error in (14) of the order 10^{6}. If equations of these magnitudes are loaded into the same $\boldsymbol{f}(\cdot)$, significant error can exist in the values of mole fractions with the equation ostensibly satisfied while the solver would strain to keep reducing the error in the enthalpy equation far beyond what is significant. IDAES and Pyomo allow us to give the first equation a scaling factor of 10^{6} and the second one of 10^{-6} to bring them to the same basis.

Let us note the effect of such scaling on the function and the Jacobian. We would replace (13) and (14) with

$$10^{6}(y_{trace,in} - y_{trace,out}) = \tilde{f}_1(\boldsymbol{x}) \tag{15}$$

$$10^{-6}(H_{in} + Q - H_{out}) = \tilde{f}_2(\boldsymbol{x}) \tag{16}$$

and the corresponding rows in the Jacobian would be replaced by

$$J(\boldsymbol{x}) \coloneqq \begin{bmatrix} 10^{6}\frac{\partial f_1}{\partial x_1} & \cdots & 10^{6}\frac{\partial f_1}{\partial x_n} \\ 10^{-6}\frac{\partial f_2}{\partial x_1} & \cdots & 10^{-6}\frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{17}$$

Now we have coefficients in the Jacobian of extremely different values. Row 1 has norm $10^{6}\sqrt{2}$ and row 2 has norm of $10^{-6}\sqrt{3}$. We can derive a lower bound for the condition number in terms of rows with largest and smallest 2-norm. Rearrange Jacobian rows so that they are sorted in descending order of 2-norm magnitude

$$J(\boldsymbol{x}) = \begin{bmatrix} r_{max} \\ \vdots \\ r_{min} \end{bmatrix} \tag{18}$$

Then we have that

$$\|r_{\max}\|_2 = \left\| \begin{bmatrix} 1 & 0 & \ldots \end{bmatrix} \begin{bmatrix} r_{max} \\ \vdots \\ r_{min} \end{bmatrix} \right\|_2 \leq \sigma_1 \tag{19}$$

and

$$\|r_{\min}\|_2 = \left\| \begin{bmatrix} \ldots & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{max} \\ \vdots \\ r_{min} \end{bmatrix} \right\|_2 \geq \sigma_n \tag{20}$$

so, we have that

$$\kappa_2\big(J(\boldsymbol{x})\big) = \sigma_1/\sigma_n \geq \|r_{\max}\|_2/\|r_{\min}\|_2 \tag{21}$$

Presently, the Jacobian matrix has condition number greater than $\frac{\sqrt{6}}{3}\cdot 10^{12}$. To reduce it, we need to also set scaling factors for variables. We define

$$\tilde{y}_{trace,in} \coloneqq 10^{6} y_{trace,in}$$
$$\tilde{H}_{out} \coloneqq 10^{-6} H_{out}$$

Suppose we had $y_{trace,in} = x_1$ and $H_{out} = x_n$. The Jacobian now looks like

$$J(\boldsymbol{x}) \coloneqq \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & 10^{12}\frac{\partial f_1}{\partial x_n} \\ 10^{-12}\frac{\partial f_2}{\partial x_1} & \ldots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{22}$$

The entry involving 10^{12} would be potentially problematic, but because $\frac{\partial f_1}{\partial x_n} = 0$, it is irrelevant. In short, scaling a constraint *multiplies* the corresponding Jacobian *row* by that scaling factor, and scaling a variable *divides* the corresponding Jacobian *column* by that scaling factor.

It is not necessary to manually create transformed variables and constraints because Pyomo allows the user to store scaling factors associated with variables and constraints as part of the model. The unscaled model can then be transformed into the scaled model before being passed to the nonlinear solver.[1] A scaled version of the model can also be retrieved for debugging purposes through a scaling transformation.

In a model with thousands of variables and constraints, however, it is difficult to keep track of which variables and constraints have been scaled, and whether the scaling factors assigned are appropriate. The IDAES diagnostics toolbox, described in [6], has the methods `display_constraints_with_extreme_jacobians` and `display_variables_with_extreme_jacobians` that iterate over Jacobian rows and columns, respectively, to display the ones with 2-norms larger and smaller than some user-specified tolerances (by default 10^{4} and 10^{-4}). These methods highlight areas of the model that require work, but the user still needs to determine appropriate values for scaling factors, as well as which variables and constraints need to be scaled. For example, rows 1 and 2 would be highlighted in the Jacobian of (17),

[1] Scaling at the time of passing the model to the solver can be done by use of the `ipopt_v2` solver in Pyomo through the `SolverFactory` function.

when it is the variables, not the constraints, that require scaling.

Scaling is often an iterative process. Once appropriate scaling factors are assigned to variables, bringing the Jacobian into the form (22), large or small entries can appear in additional rows, corresponding to other constraints, that would then need to be scaled. Finally, a large entry can correspond to a model that is *intrinsically* ill-conditioned. For example, when modeling a process with a large recycle ratio, large terms can appear in the corresponding Jacobian. Small impurities in the feed stream can become concentrated by factors of 100 or 1000 as a feature of the process. Scaling might be able to hide this feature, but it can cause a loss in accuracy and will be revealed by performing SVD of the Jacobian.

Singular Value Decomposition

Even once rows and columns with inappropriately large or small norms have been removed by scaling, the Jacobian's condition number, which can be revealed through the SVD of the Jacobian, can still be extremely large. Both σ_1 and σ_n appear directly in the expression for the condition number, so reducing σ_1 and increasing σ_n should help reduce it. In practice, σ_1 is usually reduced to a reasonable level by scaling rows and columns, so we will focus on increasing σ_n instead.

Consider the solution to a Newton step problem

$$\boldsymbol{f}(\boldsymbol{x}_k) = -J(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k \tag{23}$$

By decomposing $J(\boldsymbol{x})$ using the SVD and using that to solve (23), we obtain

$$\boldsymbol{f}(\boldsymbol{x}_k) = -U\Sigma V^T\,\Delta\boldsymbol{x}_k \tag{24}$$

$$-\mathrm{V}\Sigma^{-1}U^T\boldsymbol{f}(\boldsymbol{x}_k) = \mathrm{V}\Sigma^{-1}U^TU\Sigma V^T\,\Delta\boldsymbol{x}_k = \Delta\boldsymbol{x}_k \tag{25}$$

So, in solving the perturbed problem, the constraint residual is projected by U^T into orthogonal components in the space of the singular values, scaled by the inverse singular values, and those scaled values are projected by V into the space of variables.

Looking at (25) another way

$$-U^T\boldsymbol{f}(\boldsymbol{x}_k) = \Sigma V^T\Delta\boldsymbol{x}_k \tag{26}$$

Each column of U (row of U^T) is an orthogonal vector, associating elements of $\boldsymbol{f}(\boldsymbol{x}_k)$, i.e., particular constraints, with a singular value. Likewise, each column of V (row of V^T) is a vector associating elements of $\Delta\boldsymbol{x}_k$, i.e., particular variables, with a singular value. The SVD shows which constraints are being satisfied using which variables in a neighborhood of $\boldsymbol{x}_k$, and the sensitivity of the variables with respect to constraints.

If the smallest singular value $\sigma_n \ll 1$, a small change in certain function values requires a large change in variable values. By looking at the constraints involved in the nth left-singular vector u_n and the variables involved in nth right singular vector v_n we can attempt to determine what is causing this dysfunctional relationship. Because of the dense linear algebra involved, we expect most entries of u_n and v_n to be populated by nonzero numbers. A simple but crude method is to filter for indices that have values greater than some tolerance. We have found absolute values of 0.1 to 0.3 work well for this. If too many variables and constraints appear, raise the tolerance. If too few appear, decrease it.

Ideally, we would want the condition number of $J(\boldsymbol{x}_k)$ to be relatively small. In practice, however, we typically accomplish a condition number on the order of 10^6-10^8. That means filtering for singular values smaller than 10^{-8}-10^{-6} and attempting to remedy them. In general, there are four causes of small singular values:

1. Incorrect scaling of variables or constraints
2. Redundant equations, hinting a problem that over-specifies some variables and under-specifies others, i.e., global degeneracy
3. A local singularity, caused by evaluating $J(\boldsymbol{x})$ at a point where it locally loses rank, i.e., local degeneracy
4. Attempting to solve a problem that is inherently ill-conditioned.

Incorrect scaling is probably the most common cause of small singular values of values 10^{-12}-10^{-6}, followed by inherent ill-conditioning. For singular values smaller than 10^{-12}, the cause is typically local or global degeneracy. The next section presents a carbon capture application in which these concepts can be demonstrated.

A similar technique for identifying sets of degenerate equations is given by the Degeneracy Hunter algorithm proposed in [11]. This algorithm is also incorporated into the IDAES diagnostics toolbox. It differs from this technique by performing a QR factorization instead of an SVD, then solving a mixed integer linear program (MILP) to find an irreducible degenerate set of equations. Which algorithm is appropriate for a given problem depends on the algorithms available for a sparse QR factorization, a sparse SVD, and MILP solution. An advantage the SVD provides, however, is that it shows which combinations of variables are being used to satisfy combinations of constraints. If a degenerate constraint is removed, then another constraint must be added to constrain that combination of free variables.

CASE STUDY: POST COMBUSTION CARBON CAPTURE FLOWSHEET

A flowsheet in which CO_2 is captured from flue gas (from a ~690 MWe Natural Gas Combined Cycle power plant) by absorption into monoethanolamine (MEA) provides several examples in which these diagnostic tests were of great assistance in model refinement and increasing robustness. This flowsheet was first presented in [12], but despite the techniques used to provide robust

solutions, such as a four-stage initialization routine for the absorption and stripping columns, it remained fragile and prone to failure. It was desirable to improve convergence and numerical robustness so the flowsheet could be used for robust design optimization. The flowsheet consists of an absorption column, stripper column, lean-rich heat exchanger, and balance-of-plant equipment. It is divided into two sub-flowsheets, an absorber section and a stripper section, that are solved independently before being linked and solved together.

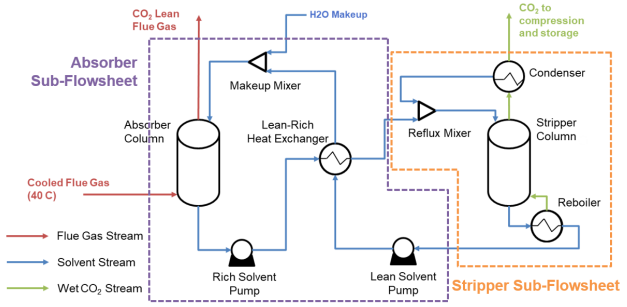

Figure 1. MEA carbon capture flowsheet, divided into stripper and absorber sections.

We begin with a flowsheet that has some partial scaling applied to the column model, but does not have scaling for any other models. The column model has already been reformulated to remove division in constraints where possible, which helps avoid bad numeric behavior when the denominator of an expression is nearly zero at an intermediate iteration. For an unscaled or partially-scaled model, the output can be hundreds of lines long. The output length is mostly the result of scaling issues in indexed variables or constraints resulting in those variables or constraints being printed for each value of their indices. Since the column model is discretized into 40 finite elements along its length, 10 badly-scaled equations results in 400 entries. Therefore, the output cannot be fully displayed here, but some representative entries can be shown.

Jacobian Analysis

Here, we run the diagnostics tools after a successful solution of the flowsheet. They can be used after failures to solve a flowsheet, but care should be taken as variables may not have realistic values, which is reflected in the model Jacobian. The following is one line (out of hundreds) produced by the `display_variables_with_extreme_jacobians` command from the `DiagnosticsToolbox`:

```
fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].temperature: 1.997E+08
```

This entry indicates a column norm of $2 \cdot 10^8$. We can inspect the associated column of the Jacobian to find which rows are associated with this large value. In IDAES-IP, we can do this using the `display_constraints_including_variable` function in the `SVDToolbox`.

```
fs.stripper_section.reflux_mixer.enthalpy_mixing_equations[0.0]: 1.997e+08

fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].log_k_eq_constraint[bicarbonate]: 5.364e+00

fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].log_k_eq_constraint[carbamate]: 6.551e+00

fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].log_conc_mol_phase_comp_true_eq[Liq,MEACOO_-]: 1.514e+00

fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].log_conc_mol_phase_comp_true_eq[Liq,HCO3_-]: 2.333e-01

fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].log_conc_mol_phase_comp_true_eq[Liq,MEA_+]: 1.748e+00

fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].log_conc_mol_phase_comp_true_eq[Liq,H2O]: 3.183e+01

fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].log_conc_mol_phase_comp_true_eq[Liq,MEA]: 1.129e+00

fs.stripper_section.reflux_mixer.rich_solvent_state[0.0].log_conc_mol_phase_comp_true_eq[Liq,CO2]: 5.685e-03

    fs.rich_temperature: 1.000e+02
```

From inspection, the problematic Jacobian entry is associated with a mixer enthalpy mixing equation that is not yet scaled.

To fix the problem, we need to determine an appropriate scaling factor. We can calculate such a value from scaling factors for both the molar flow and the specific molar enthalpy. Finding the molar flow scaling factor is easy; it is generally observed that the molar flow rates are on the order of 10^3-10^4 throughout the flowsheet, so we choose $3 \cdot 10^{-4}$ as a scaling factor for the molar flow rate. Determining a scaling factor for the specific molar enthalpy is harder. Enthalpy can be either positive or negative in different areas of the flowsheet, so its order of magnitude does not make a good scaling factor. By evaluating the liquid-phase enthalpy in the reflux mixer, the water makeup mixer, and the reboiler, we get values of -42200, -43600, and -38800. With an apparent range of 4800, a scaling factor of $3 \cdot 10^{-4}$ is also a good choice for this variable. Values for scaling factors do not have to be exact: almost counts in horseshoes, hand grenades, and model scaling.

The process of removing this scaling issue took us through at least three Pyomo sub-models (and associated Python files) to determine a process through which the scaling factor for one equation was calculated. However, had reasonable default scaling factors for the property sub-model been set by the user ahead of time, this issue would never have arisen.

Next, we consider scaling the equations describing

heat transfer in the column

$$h_V a_e \, Ack = -(\underline{C}_{pCO_2,V} N_{CO_2,V} + \underline{C}_{pH_2O,V} N_{H_2O,V}) \quad (28)$$

$$h'_V a_e (1 - \exp(Ack)) = \underline{C}_{pCO_2,V} N_{CO_2,V} + \underline{C}_{pH_2O,V} N_{H_2O,V} \quad (29)$$

in which h_V is the heat transfer coefficient from liquid to vapor, a_e is the effective area of heat transfer per unit column volume, $\underline{C}_{p,i,V}$ is the heat capacity of species i, $N_{i,V}$ is the molar flux of species i from vapor to liquid, and Ack is the Ackerman factor, which is a measure of how much diffusive heat fluxes are distorted by convective heat transfer. It was discovered that $|Ack| \leq 10^{-2}$ for operating conditions of interest and was frequently on the order of 10^{-5}. This causes ill-conditioning in both equations because the factors Ack and $1 - \exp(Ack)$ became close to zero. Because Ack could vary over several orders of magnitude depending on location and operating conditions, assigning a consistent scaling factor is difficult.

If we first rearrange (28) and (29) to make the relationship between h_V and h'_V more clear

$$h'_V = h_V \frac{Ack}{(\exp(Ack) - 1)} \quad (30)$$

several solutions are possible. The function

$$\theta(x) = \frac{x}{\exp(x) - 1} \quad (31)$$

has an indeterminant form at $x = 0$, but a well-defined Taylor expansion

$$\theta(x) \approx 1 - \frac{x}{2} + \frac{x^2}{12} + O(x^4) \quad (32)$$

A Taylor approximation could be substituted for $\theta(Ack)$ in (30), but $\theta(x)$ could also be implemented as an external grey-box function in IDAES, switching between the full form (31) and Taylor form (32) based on $|x|$. Both options were eventually implemented. An external function for $\theta(x)$ is implemented in IDAES with a sixth-order Taylor approximation, but the trivial Taylor approximation $\theta(x) \approx 1$ is presently used in the MEA flowsheet because not every solver in Pyomo supports external functions.

SVD Analysis

The SVD also tells us valuable information about the state of the flowsheet. Its condition number is $9.2 \cdot 10^{17}$, so the matrix is singular to machine precision. The smallest singular value is $\sigma_n = 3.1 \cdot 10^{-10}$. Using a tolerance for the singular vectors of 0.1, we find the following variables involved:

```
fs.stripper_section.reflux_mixer.reflux_state[0.0].log_conc_mol_phase_comp_true[Liq,HCO3_-]
```

```
fs.stripper_section.reflux_mixer.reflux_state[0.0].log_conc_mol_phase_comp_true[Liq,MEA_+]
```

```
fs.stripper_section.reflux_mixer.reflux_state[0.0].log_conc_mol_phase_comp_true[Liq,MEA]
```

```
fs.stripper_section.reflux_mixer.reflux_state[0.0].log_conc_mol_phase_comp_true[Liq,MEACOO_-]
```

and the following constraints involved:

```
fs.stripper_section.condenser.liquid_phase[0.0].appr_to_true_species[Liq,MEA]
```

```
fs.stripper_section.condenser.liquid_phase[0.0].true_mole_frac_constraint[Liq,HCO3_-]
```

```
fs.stripper_section.condenser.liquid_phase[0.0].true_mole_frac_constraint[Liq,MEA_+]
```

```
fs.stripper_section.condenser.liquid_phase[0.0].true_mole_frac_constraint[Liq,MEA]
```

```
fs.stripper_section.condenser.liquid_phase[0.0].log_conc_mol_phase_comp_true_eq[Liq,HCO3_-]
```

```
fs.stripper_section.condenser.liquid_phase[0.0].log_conc_mol_phase_comp_true_eq[Liq,MEA_+]
```

```
fs.stripper_section.condenser.liquid_phase[0.0].log_conc_mol_phase_comp_true_eq[Liq,MEA]
```

```
fs.stripper_section.condenser.liquid_phase[0.0].appr_to_true_species[Liq,HCO3_-]
```

```
fs.stripper_section.condenser.liquid_phase[0.0].appr_to_true_species[Liq,MEA_+]
```

All these variables and constraints occur in the condenser, which makes interpretation of the problem easier. However, if multiple degeneracies are present, they can mix between different tiny singular values, so sometimes additional analysis is necessary to separate different degeneracies.

Because the property model in this flowsheet does not account for amine volatility, the mole fraction of MEA in the condenser is effectively zero. All the variables implicated here are dissociation species of MEA, which similarly have concentrations of effectively zero. The constraints are likewise the disassociation equations for MEA. Thus, the absence of MEA causes degeneracy in the system of equations governing the dissociation reactions. Different strategies can be employed to overcome this degeneracy. The easiest is to increase the mole fraction of MEA to around 10^{-4}. However, this merely mitigates the ill-conditioning—it does not remove it—and the addition of extra MEA in the system could cause problems with material balances converging.

IDAES-IP allows the user to define property packages that bundle together thermodynamic calculations into a single sub-model that can then be employed in different unit models. The solution we employed was to create a duplicate property package without the dissociation reactions and use that for the condenser and reflux mixer. That solution works only because the liquid phase property sub-model does not rely on ion concentrations for the calculation of enthalpy. For one that requires ion concentrations to calculate the enthalpy of mixing, like

eNRTL, another solution would have to be devised.

Limits of Scaling and Reformulation

Not every problem that can be discovered by these diagnostic tools has a nice solution. Such is the case with the system of equations for the enhancement factor for mass transfer in reactive systems. Taken from [13] for the MEA-CO_2 system, the full enhancement factor model has ten tightly coupled numerical expressions in it. However, the core numerical issues derive from two equations:

$$E = 1 + (E_\infty^* - 1)\frac{1-\Upsilon_{MEA}^i}{1-\Upsilon_{CO_2}^b} \tag{33}$$

$$E = Ha\sqrt{\Upsilon_{MEA}^i}\frac{1-\Upsilon_{CO_2}^*}{1-\Upsilon_{CO_2}^b} \tag{34}$$

in which E is the enhancement factor, E_∞^* is the "instantaneous enhancement factor," a theoretical maximum value for the enhancement factor, Υ_{MEA}^i is a dimensionless concentration of MEA at the vapor/liquid interface, $\Upsilon_{CO_2}^*$ is the dimensionless concentration of CO_2 at the interface, $\Upsilon_{CO_2}^b$ is the dimensionless concentration of CO_2 in the bulk liquid, and Ha is the Hatta number, the ratio of reaction film to the rate of diffusion in the film. Absorption happens when $\Upsilon_{CO_2}^b < 1$, desorption happens when $\Upsilon_{CO_2}^b > 1$, and equilibrium occurs when $\Upsilon_{CO_2}^b = 1$.

The problem occurs near equilibrium, because the expression $1 - \Upsilon_{CO_2}^b$ is nearly equal to zero, and the quotients in (33) and (34) are nearly singular. At actual solutions to the system of equations, numerical tests show that $1 - \Upsilon_{CO_2}^*$ and $1 - \Upsilon_{MEA}^i$ also approach zero, so the quotient remains defined. However, since we now must deal with a multivariate function, we have been unsuccessful at removing the singularity by use of a Taylor series, like we did with the Ackerman factor. The expressions $1 - \Upsilon_{CO_2}^b$ and $1 - \Upsilon_{CO_2}^*$ will have the same sign at any solution to the activity factor system. However, contrary to what is stated in [13], there exist at least one scenario in which $1 - \Upsilon_{CO_2}^b$ has the opposite sign as $1 - \Upsilon_{MEA}^i$. When CO_2 is desorbing at low temperatures, the rate of reaction can become slow enough that the rate of desorption is lower with reaction than it would be without reaction, and the enhancement factor drops below one. That fact limits possible reformulations of these two equations.

The equations were ultimately reformulated in several steps. First, a log variable S_{CO_2} was introduced for one of the singular ratios:

$$(1 - \Upsilon_{CO_2}^b)\exp(S_{CO_2}) = 1 - \Upsilon_{CO_2}^* \tag{35}$$

so that (34) could be rewritten as the linear equation

$$\log E = \log Ha + \frac{1}{2}\log \Upsilon_{\mathrm{MEA}}^i + S_{CO_2} \tag{36}$$

in which $\log E$, $\log Ha$, and $\log \Upsilon_{\mathrm{MEA}}^i$ are all additional log variables defined using equations of the form

$$\exp(\log x) = x \tag{37}$$

To be completely clear, $\log x$ is a single variable, not an expression denoting the logarithm of the variable x, which we denote $\log(x)$. We have that $\log x = \log(x)$ at solutions of the model, i.e., when $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, but equality does not necessarily hold at intermediate Newton iterates. The benefit of implicitly taking the logarithm in equations like (35) and (37) instead of using a log function in (36) is that the variable $\log x$ can maintain numerical accuracy even when the values of x are extremely small or extremely large.

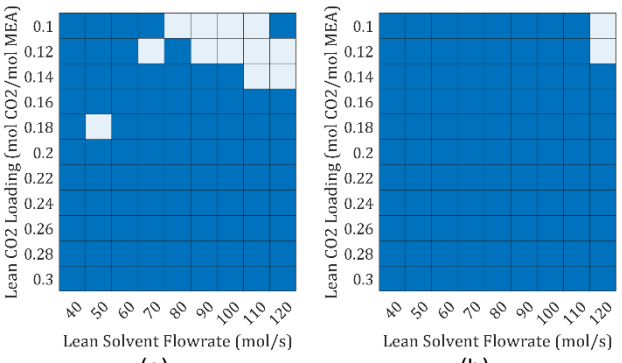

Figure 1. Convergence of the pilot-scale stand-alone column model depending on inlet parameters **(a)** before and **(b)** after the scaling and reformulations detailed here. A filled square indicates it converged, an unfilled square indicates it did not converge. In the reformulated model, divergence typically is a result of $\Upsilon_{CO_2}^*$ incorrectly converging to 1, resulting in (35) and (38) becoming degenerate.

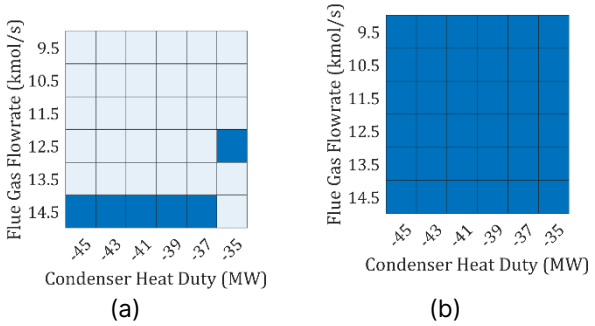

Figure 2. Convergence of the plant-scale full flowsheet model depending on inlet parameters **(a)** before and **(b)** after the scaling and reformulations detailed here. A filled square indicates it converged, an unfilled square indicates it did not converge.

Unfortunately, both (35) and (39) become degenerate when $\Upsilon_{CO_2}^b = 1$. Because the enhancement factor system can be largely decoupled from the remaining equations, a surrogate model was created to

give the log enhancement factor as a function of liquid phase CO_2 and H_2O loading, vapor phase CO_2 partial pressure, temperature, and liquid and vapor mass transfer coefficients. However, too much accuracy was lost in the first attempt at a surrogate model, and the reformulated enhancement factor model is reliable enough to keep. Figures 1a and 1b show the convergence of the original and reformulated stand-alone column models, while Figures 2a and 2b show the convergence of the original and reformulated full flowsheet models (with all changes, not just to the enhancement factor calculations), respectively. The numerical robustness of the reformulated and scaled models is significantly improved over that of the original model.

It is possible to use the same technique as (35) for the ratio of MEA to CO_2 in (33). However, that implicitly assumes that $1-\Upsilon^{*}_{CO_2}$ and $1-\Upsilon^{i}_{MEA}$ have the same sign, which is true for most operating conditions of interest, but is not true for desorption at low temperatures. The resulting reformulation of (33) is then

$$\exp(\log E)(1-\Upsilon^{b}_{CO_2}) = \exp(\eta)\left(1-\Upsilon^{i}_{MEA}\right) \quad (38)$$

$$\exp(\eta) = E^{*} - 1 \quad (39)$$

in which the log variable η has been introduced for the instantaneous enhancement factor minus one because there is a simple expression for η (not given here) in terms of logarithms of other variables.

CONCLUSIONS

We have outlined model diagnostic methods using tools available in Pyomo and IDAES-IP. As illustrated through the example of the MEA flowsheet, these tools can be of great assistance to the user by serving to point out likely problems. However, they do not solve them for the user. Finding problems in a model with tens of thousands of constraints and variables is a major service, but the user's insight and modelling expertise is still necessary to solve them. Nevertheless, these tools and techniques can help make EO modeling frameworks more accessible and facilitate the use of advanced optimization techniques in process design.

Acknowledgements

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 through the Simulation-based Engineering/Crosscutting Research and Carbon Capture Programs.

Disclaimer

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, expressor 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. US Department of Energy. https://www.energy.gov/articles/how-were-moving-net-zero-2050
2. 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:e10095. (2021)
3. 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. Springer (2021).
4. Hart, W.E., Watson, J.-P., Woodruff, D.L. Pyomo: modeling and solving mathematical programs in Python. *Math Program Comput* 3 219–260. (2011)
5. ASPENTech. ASPEN Engineering Suite.
6. Lee, A., Dowling, A. W., Parker, R., Poon, S., Gunter, D., Nicholson, B. Model Diagnostics for Equation Oriented Models: Roadblocks and the Path Forward. *Proc of FOCAPD*. (2024)
7. Gay, D.M., Hooking Your Solver to AMPL. (2017)
8. Rodriguez, J.S., Parker, R.B., Laird, C.D., Nicholson, B.L., Siirola, J.D., Bynum, M.L. Scalable Parallel Nonlinear Optimization with PyNumero and Parapint. *INFORMS J Comput* 35:509–517. (2023)
9. Wächter, A., Biegler, L.T. On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. *Math Program* 106:25-57. (2004)
10. Braatz, R.D., Morari, M. Minimizing the Euclidean Condition Number. *Siam J Control Opt* 32:1763–1768 (1994)
11. Dowling A.W., Biegler L.T. (2015). Degeneracy Hunter: An Algorithm for Determining Irreducible

Sets of Degenerate Constraints in Mathematical Programs. *Comput Aided Chem Eng.* 37:809 –814 (2015)
12. Akula, P., Eslick, J., Bhattacharyya, D., Miller, D.C. Model Development, Validation, and Optimization of an MEA-Based Post-Combustion CO_2 Capture Process under Part-Load and Variable Capture Operations. *Ind Eng Chem Res* 60:5176–5193 (2021)
13. Gaspar, J., Fosbøl, P.L. A general enhancement factor model for absorption and desorption systems: A CO2 capture case-study. *Chem Eng Sci* 138:203–215 (2015)

© 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/