An implicit function formulation for optimization of discretized index-1 differential algebraic systems

Robert Parker, Bethany Nicholson, John Siirola, Carl Laird, and Lorenz Biegler

March 2022

Abstract

A formulation for the optimization of index-1 differential algebraic equation systems (DAEs) that uses implicit functions to remove algebraic variables and equations from the optimization problem is described. The formulation uses the implicit function theorem to calculate derivatives of functions that remain in the optimization problem in terms of a reduced space of variables, allowing it to be solved with second-order nonlinear optimization algorithms. The formulation is shown to lead to more reliable solver convergence when compared with a full-space simultaneous formulation in challenging case studies involving a chemical looping combustion reactor. In a steady state simulation problem, the implicit function formulation solves 82 out of 100 instances, while the full space formulation solves 42 out of 100 instances. In a dynamic optimization problem, the implicit function formulation solves 189 out of 200 instances, while the full space formulation solves 129 out of 200 instances. In these examples, the nonlinear system of algebraic equations is maintained feasible throughout the optimization, reducing the likelihood that poor conditioning of these equations will cause non-convergence. In bound-constrained instances of the dynamic optimization problem, the full space formulation converges in an average of 83 CPU s while the implicit function formulation converges in an average of 64 CPU s. For the simulation problem, however, the implicit function formulation will require a parallel implementation to be competitive with the full space formulation. On the other hand, our results indicate the potential for the reduced space formulation to be more reliable than the full space formulation for challenging nonlinear DAE optimization problems, without appreciable performance degradation for large problems.

1 Introduction

Differential algebraic equations (DAEs) are an expressive class of equations for chemical and process systems engineers. They are capable of representing processes that evolve over continuous time and/or space domains and involve nonlinear physical and chemical phenomena. Complicated interactions among differential states are often described by large numbers of highly nonlinear algebraic equations that represent the thermodynamics, chemical reactions, and transport phenomena of the system. Due to their number and complexity, the ability to simulate or optimize a large-scale DAE relies heavily on the ability to converge these algebraic equations.

Optimization of nonlinear DAE systems is typically done by solving nonlinear programming (NLP) problems, as local solvers for these problems are mature and capable of handling

large scale systems. Two common approaches for these optimization problems are the sequential approach described by Goh & Teo (1988) and the simultaneous approach described by Cuthrell & Biegler (1987). In the former, the optimization algorithm is only aware of control inputs and computes the objective function by simulating a DAE system. Gradient information is returned by the DAE simulator and used to update the input variables. While many applications of sequential dynamic optimization use only first-order derivative information, Vassiliadis et al. (1999) and Ozyurt & Barton (2005) calculate second derivatives with respect to control inputs, and Hannemann & Marquardt (2007) apply these calculations to an optimal control problem.

Hybrid approaches to DAE optimization have been proposed, including the multiple shooting algorithm of Bock & Plitt (1984), which partitions the continuous domain and simulates multiple DAEs sequentially, and the quasi-sequential approach of Hong et al. (2006), which eliminates all equality constraints from the optimization algorithm but retains inequality constraints that may involve state variables. In addition, Kouzoupis et al. (2015) propose a block-condensing algorithm that uses multiple shooting with a specialized sequential quadratic programming (SQP) solver for structured subproblems at each set of time points.

Sequential dynamic optimization, including hybrid approaches, may be viewed as optimization problems with embedded implicit functions, similar to sequential and simultaneous modular approaches for flowsheet optimization (Kisala et al. (1987)). In turn, optimization with embedded implicit functions may be viewed as a class of two-level decomposition approaches such as the alternating direction method of multipliers (ADMM) (Eckstein & Bertsekas (1992)) and the approach of Yoshio & Biegler (2021) for multiperiod optimization. In the context of local optimization, these strategies may exhibit more reliable convergence than fully simultaneous approaches, although more expensive computations are performed at every iteration of the optimization algorithm. Large computation times may be mitigated, however, if lower-level problems are solved in parallel, as in Yoshio & Biegler (2021).

Optimization with embedded implicit functions has also received recent attention in the global optimization field, for example in the studies of Bongartz & Mitsos (2017) and Wilhelm et al. (2019), which were made possible by the advances of Mitsos et al. (2009) and Stuber et al. (2015).

In contrast to fully sequential dynamic optimization, we propose a hybrid approach for DAEs that are index-1 in which only the algebraic variables and equations are removed from the nonlinear optimization problem using implicit functions. Our approach is simultaneous in the sense that the continuous domain is discretized and that all discretization points are considered simultaneously by the optimizer, but is also similar to sequential approaches as square problems are solved by an embedded solver within each iteration of the optimization solve. This is a reduced-space formulation as only the differential, input, and derivative variables are seen by the optimization algorithm. The implicit function subproblems admit exact first and second derivatives in terms of these variables and are thus compatible with second-order optimization algorithms.

Using implicit functions to solve optimization problems in a reduced space has several advantages. While simultaneous dynamic optimization formulations and nonlinear optimization solvers are relatively mature, their convergence is sensitive to an initial guess and to scaling of variables and constraints. We note that a reduced space formulation leaves a user with fewer variables to initialize and scale. In addition, in some cases, the difficulty of converging a nonlinear optimization problem may be due to the effort required to converge a large system of nonlinear equality constraints. If these equations are solved

in a smaller system separate from the optimization, however, they are more likely to converge. In addition, at iterates that are feasible with respect to the algebraic equations, the optimization algorithm is less likely to encounter severe ill-conditioning that may lead to non-convergence. For these reasons, an implicit function formulation is likely to improve the ease and reliability of solver convergence.

2 Background

This work is concerned with the optimization of an index-1 DAE in a simultaneous (fully discretized) formulation.

A DAE has the form given by Equation (1), where (1a) are the differential equations, (1b) are the algebraic equations, and (1c) are the initial conditions. Vector x is a vector of differential variables, y is a vector of algebraic variables, u is a vector of manipulated inputs, and $\dot{x}$ is the vector of time derivatives of x. Functions F and G are vector-valued differential and algebraic functions that are assumed twice continuously differentiable in all arguments. All variables are functions of time, denoted by the subscript t. In this continuous DAE, t takes values between t_0 and t_f.

$$\dot{x}_t = F(x_t, y_t, u_t) \qquad \text{for all } t \text{ in } [t_0, t_f] \tag{1a}$$
$$0 = G(x_t, y_t, u_t) \qquad \text{for all } t \text{ in } [t_0, t_f] \tag{1b}$$
$$x_{t_0} = \bar{x}_0 \tag{1c}$$

The discretized DAE is obtained by applying Equations (1a) and (1b) only at discrete time points by adding discretization equations to approximate the time derivatives $\dot{x}$. Equation (2) shows the equations for the discretized DAE. For an implicit discretization, the discretization equation (2c) is omitted at t_0.

$$\dot{x}_t = F(x_t, y_t, u_t) \qquad \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \tag{2a}$$
$$0 = G(x_t, y_t, u_t) \qquad \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \tag{2b}$$
$$0 = d(x_{t_i}, x_{t_{i-1}}, \dot{x}_{t_i}) \qquad \text{for all } t_i \text{ in } \{t_1, \ldots, t_f\} \tag{2c}$$
$$x_{t_0} = \bar{x}_0 \tag{2d}$$

A DAE is index-1 if the Jacobian of the function defining the algebraic equations with respect to algebraic variables, $\nabla_y G$, is nonsingular for all values of differential, algebraic, and input variables. In this case, by the implicit function theorem, there exists a function G_y which maps the vector (x, u) to y such that Equation (2b) is satisfied.

A simultaneous nonlinear programming formulation for the optimization of a discretized DAE model is given in Equation (3). This is referred to as the full space formulation. Here, φ is a scalar-valued cost function and X and U are sets representing upper and lower bounds on differential variables and control inputs. A dynamic optimization problem of this form can be constructed in an algebraic modeling environment such as Pyomo (Hart et al. (2011)) if the constraints and objective are explicit functions of the variables. The optimization problem may then be solved with a generic nonlinear programming solver such

as IPOPT (Wächter & Biegler (2006)).

$$\begin{aligned}
\min_{(\dot{x},x,y,u)} \quad & \sum_{t=t_0}^{t_f} \varphi(x_t, u_t) \\
\text{s.t.} \quad & \dot{x}_t - F(x_t, y_t, u_t) = 0 && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& G(x_t, y_t, u_t) = 0 && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& d(x_{t_i}, x_{t_{i-1}}, \dot{x}_{t_i}) = 0 && \text{for all } t_i \text{ in } \{t_1, \ldots, t_f\} \\
& x_{t_0} = \bar{x}_0 \\
& x_t \in X, \;\; u_t \in U && \text{for all } t \text{ in } \{t_0, \ldots, t_f\}
\end{aligned} \tag{3}$$

Our implicit function formulation takes advantage of the index-1 property to eliminate algebraic variables and equations from the NLP. That is, (2b) and the index-1 property imply that $y = G_y(x, u)$, where G_y is a (locally defined) implicit function. The implicit function formulation of the discretized DAE is given in Equation (4).

$$\begin{aligned}
\min_{(\dot{x},x,u)} \quad & \sum_{t=t_0}^{t_f} \varphi(x_t, u_t) \\
\text{s.t.} \quad & \dot{x}_t - F(x_t, G_y(x_t, u_t), u_t) = 0 && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& d(x_{t_i}, x_{t_{i-1}}, \dot{x}_{t_i}) = 0 && \text{for all } t_i \text{ in } \{t_1, \ldots, t_f\} \\
& x_{t_0} = \bar{x}_0 \\
& x_t \in X, \;\; u_t \in U && \text{for all } t \text{ in } \{t_0, \ldots, t_f\}
\end{aligned} \tag{4}$$

An optimization problem of this form cannot be constructed directly in an algebraic modeling environment because G_y is an implicit function. This formulation can still be solved by a nonlinear programming algorithm, however, if G_y may be evaluated and admits sufficient derivative information. We implement an interface that evaluates an implicit function and computes its first and second derivatives via the implicit function theorem. Our implementation is described in Section 3. These values and derivatives can be communicated to any NLP solver with a direct interface available, allowing us to solve DAE optimization problems in the form of Equation (4).

3 Implicit function formulation

The implicit function formulation addresses NLPs of the form shown by Equation (5). Here (5b) are referred to as residual equations, which are exposed to the NLP solver, while (5c) are referred to as external equations, which are removed as an implicit function along with the external variables b, which have dimension n_b. Vector a, with dimension n_a, is referred to as the vector of internal variables. Function f has outputs of dimension n_f, g has outputs of dimension n_g, which is assumed to be equal to n_b, and φ is scalar-valued.

$$\min_{(a,b)} \quad \varphi(a, b) \tag{5a}$$

$$\text{s.t.} \quad f(a, b) = 0 \tag{5b}$$

$$g(a, b) = 0 \tag{5c}$$

$$a \geq 0 \tag{5d}$$

Assuming the Jacobian of external equations with respect to external variables, $\nabla_b g$ is nonsingular, Equation (5c) may be removed from the NLP formulation and used to solve for b as a function of a. The system seen by the NLP solver is given by Equation (6).

$$\min_a \ \bar{\varphi}(a) = \varphi(a, b(a)) \tag{6a}$$

$$\text{s.t.} \ \ \bar{f}(a) = f(a, b(a)) = 0 \tag{6b}$$

$$a \geq 0 \tag{6c}$$

To solve this NLP with a second-order optimization algorithm, we need the first and second derivatives of $\bar{f}$ and $\bar{\varphi}$. The Jacobians $\nabla_a \bar{f}$ and $\nabla_a \bar{\varphi}$ are used in the matrix and right-hand-side of the Karush-Kuhn-Tucker (KKT) system. The Hessians $\nabla^2_{aa} \bar{f}$ and $\nabla^2_{aa} \bar{\varphi}$ are used in the Hessian of the Lagrangian of (6), $\nabla^2_{aa} \bar{\mathcal{L}}$. The calculation of these matrices via the implicit function theorem is detailed in Section 3.1.

This formulation easily encapsulates the index-1 simultaneous DAE formulations (3) and (4). External variables and equations are the algebraic variables and equations at every point in time, while residual equations are the differential equations and initial conditions. The condition required to apply the implicit function formulation, nonsingularity of $\nabla_b g$, is the same as the index-1 property assumed for the DAE model.

3.1 Derivative calculations

To solve the implicit function NLP formulation (6) with a second-order optimization algorithm, we need to implement routines to calculate the constraint Jacobian $\nabla_a \bar{f}$, the objective gradient $\nabla_a \bar{\varphi}$, and the Hessian of the Lagrangian $\nabla^2_{aa} \bar{\mathcal{L}}$ given values of a and the equality multiplier vector $\bar{\lambda}$. These derivatives can be expressed in terms of f and g and their derivatives with respect to a and b.

By the implicit function theorem, the Jacobian of b as a function of a is given by Equation (7). Then by the chain rule, the Jacobian of $\bar{f}$ is given by Equation (8) and the gradient of $\bar{\varphi}$ is given by Equation (9). We use the convention that, in a Jacobian matrix, rows correspond to variables and columns correspond to constraints.

$$\nabla_a b^T = -\nabla_b g^{-T} \nabla_a g^T \tag{7}$$

$$\nabla_a \bar{f}^T = \nabla_a f^T + \nabla_b f^T \nabla_a b^T \tag{8}$$

$$\nabla_a \bar{\varphi} = \nabla_a \varphi + \nabla_a b \nabla_b \varphi \tag{9}$$

The Lagrangian of (6) is given by Equation (10), in which $\bar{\lambda}$ is the vector of equality constraint multipliers and z is the vector of bound multipliers. For brevity, arguments of functions are omitted. Functions are always evaluated at values of a specified by the optimization algorithm and the corresponding value of b obtained by solving $g(a, b) = 0$.

The gradient of the Lagrangian, or its partial derivative with respect to a, is given by Equation (11), where i indexes the coordinates of $\bar{f}$ and $\nabla_a \bar{f}_i$ is the i^{th} column of the derivative matrix. Equation (11b) uses (8) and (9) to substitute for $\nabla_a \bar{\varphi}$ and $\nabla_a \bar{f}$.

$$\bar{\mathcal{L}} = \varphi + \bar{\lambda}^T \bar{f} + z^T a \tag{10}$$

$$\nabla_a \bar{\mathcal{L}} = \nabla_a \bar{\varphi} + \sum_i \bar{\lambda}_i \nabla_a \bar{f}_i + z \tag{11a}$$

$$\nabla_a \bar{\mathcal{L}} = \nabla_a \varphi + \nabla_a b \nabla_b \varphi + \sum_i \bar{\lambda}_i \left(\nabla_a f_i + \nabla_a b \nabla_b f_i \right) + z \tag{11b}$$

The Hessian of the Lagrangian with respect to a, $\nabla^2_{aa} \bar{\mathcal{L}}$, is given by Equation (12), where λ_g is given by Equation (13). Here, j indexes the coordinates of g, and $\nabla^2_{ab} g_j$ is the $n_a \times n_b$ Hessian matrix of the j^{th} coordinate of g. This expression is obtained by differentiation of both sides of Equation (11b). A detailed derivation is provided in Appendix A.

$$\begin{aligned}
\nabla^2_{aa} \bar{\mathcal{L}} =& \nabla^2_{aa} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{aa} f_i + \sum_j \lambda_{g_j} \nabla^2_{aa} g_j \\
&+ \left(\nabla^2_{ab} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{ab} f_i + \sum_j \nabla^2_{ab} g_j \lambda_{g_j} \right) \nabla_a b^T + \nabla_a b \left(\nabla^2_{ba} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{ba} f_i + \sum_j \nabla^2_{ba} g_j \lambda_{g_j} \right) \\
&+ \nabla_a b \left(\nabla^2_{bb} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{bb} f_i + \sum_j \lambda_{g_j} \nabla^2_{bb} g_j \right) \nabla_a b^T
\end{aligned} \tag{12}$$

$$\lambda_g = -\nabla_b g^{-1} \left(\nabla_b f \bar{\lambda} + \nabla_b \varphi \right) \tag{13}$$

The form of the Hessian provided here is chosen for computational efficiency. A procedure to calculate this derivative is:

1. Evaluate all first derivatives of f, g, and φ
2. Factorize $\nabla_b g$
3. Compute $\nabla_a b$ via (7)
4. Compute λ_g via (13)
5. Compute $\nabla^2_{aa} \bar{\mathcal{L}}$ via (12)

Note that calculation of λ_g and multiplications by $\nabla_a b$ occur outside the summations over i and j and thus are not unnecessarily repeated when this expression is coded. Several terms in this formulation also resemble the Hessian of the Lagrangian of the full space problem (5). The relationship between the Hessian of the Lagrangian in the full and reduced space formulations is given in Section 3.3. With the ability to calculate the gradients of $\bar{f}$ and $\bar{\varphi}$ and the Hessian of the Lagrangian at any values of a and $\bar{\lambda}$, the implicit function NLP formulation (6) may be solved with a second-order nonlinear optimization algorithm such as IPOPT.

3.2 Implementation

The implicit function formulation is implemented in Pyomo (Bynum et al. (2021)) using PyNumero (Rodriguez et al. (2021)) and the AMPL Solver Library (Gay (1997)) for derivative calculations in Equations (8) and (12). The implicit function is evaluated by solving $g(a, b) = 0$ with a Newton solver that exploits block triangular structure via the procedure described by Duff & Reid (1978). The optimization problem is solved by IPOPT (Wächter & Biegler (2006)) via the direct Python interface provided by CyIpopt.

3.3 Relationship to full space linear system

The symmetric KKT system, solved at each iteration of interior point solvers such as IPOPT, is given in Equation (14) for the implicit function NLP formulation. Here, A is the diagonal matrix of primal variables a, Z is the diagonal matrix of bound multipliers, d_a is the search direction in the primal variables, d_λ is the search direction in the equality multipliers, μ is the barrier parameter, and e is a vector of all ones. Matrices $\nabla^2_{aa}\bar{\mathcal{L}}$ and $\nabla_a \bar{f}$ are given by Equations (12) and (8). Function arguments are omitted for brevity. All functions are evaluated at the current primal iterate a and, where applicable, the corresponding b vector obtained by solving $g(a, b) = 0$.

$$\begin{bmatrix} \nabla^2_{aa}\bar{\mathcal{L}} + A^{-1}Z & \nabla_a \bar{f} \\ \nabla_a \bar{f}^T & \end{bmatrix} \begin{pmatrix} d_a \\ d_\lambda \end{pmatrix} = - \begin{pmatrix} \nabla_a \bar{\varphi} + \nabla_a \bar{f} d_\lambda + A^{-1}\mu e \\ \bar{f} \end{pmatrix} \tag{14}$$

The symmetric KKT system in the full space formulation is given by Equation (15), where the Hessian of the Lagrangian of the full space formulation is given by Equation (16) in terms of partial derivatives. Vectors λ_g and λ_f contain equality constraint multipliers with respect to constraints (5b) and (5c) respectively.

$$\begin{bmatrix} \nabla_{aa}\mathcal{L} + A^{-1}Z & \nabla_{ab}\mathcal{L} & \nabla_a f & \nabla_a g \\ \nabla_{ba}\mathcal{L} & \nabla_{bb}\mathcal{L} & \nabla_b f & \nabla_b g \\ \nabla_a f^T & \nabla_b f^T & & \\ \nabla_a g^T & \nabla_b g^T & & \end{bmatrix} \begin{pmatrix} d_a \\ d_b \\ d_{\lambda_f} \\ d_{\lambda_g} \end{pmatrix} = - \begin{pmatrix} \nabla_a \varphi + \nabla_a f \lambda_f + \nabla_a g \lambda_g + A^{-1}\mu e \\ \nabla_b \varphi + \nabla_b f \lambda_f + \nabla_b g \lambda_g \\ f \\ g \end{pmatrix} \tag{15}$$

$$\begin{aligned} \nabla^2_{aa}\mathcal{L} &= \nabla^2_{aa}\varphi + \textstyle\sum_i \lambda_{f_i} \nabla^2_{aa} f_i + \sum_j \lambda_{g_j} \nabla^2_{aa} g_j \\ \nabla^2_{ab}\mathcal{L} &= \nabla^2_{ab}\varphi + \textstyle\sum_i \lambda_{f_i} \nabla^2_{ab} f_i + \sum_j \lambda_{g_j} \nabla^2_{ab} g_j \\ \nabla^2_{bb}\mathcal{L} &= \nabla^2_{bb}\varphi + \textstyle\sum_i \lambda_{f_i} \nabla^2_{bb} f_i + \sum_j \lambda_{g_j} \nabla^2_{bb} g_j \end{aligned} \tag{16}$$

This section shows that, given the same internal variables a and multipliers λ_f (λ in the implicit function formulation), these two linear systems calculate the same search direction in a and λ_f if the full space system is evaluated at a point of zero primal and dual infeasibility in the coordinates of variables b and constraints (5c). This assumption means that the second and fourth blocks of the right-hand side of (16) are zero, i.e.

$$g(a, b) = 0 \tag{17}$$

$$\nabla_b \varphi + \nabla_b f \lambda_f + \nabla_b g \lambda_g = 0 \tag{18}$$

$$\lambda_g = -\nabla_b g^{-1} \left(\nabla_b \varphi + \nabla_b f \lambda_f \right) \tag{19}$$

Equation (17) and the nonsingularity of $\nabla_b g$ allow the implicit function theorem to hold, implying that b may be expressed as a (local) function of a, with derivative $\nabla_a b$ given by Equation (7). Similarly, we may use Equation (18) to obtain λ_g as a function of λ_f, given by Equation (19). We note that λ_g is the same as defined in the implicit function Lagrangian Hessian by Equation (13), now with λ_f instead of $\bar{\lambda}$.

By pivoting (15) on the second element of the last block row, now with right-hand side of zero, we obtain Equation (20) and use it to eliminate the second block column of the KKT matrix. The resulting system is Equation (21), where L_μ is given by Equation (22) in which λ_g has been substituted using Equation (19).

$$d_b = -\nabla_b g^{-T} \nabla_a g^T d_a \tag{20}$$

$$\begin{bmatrix} \nabla_{aa}\mathcal{L} + A^{-1}Z - \nabla_{ab}\mathcal{L}\nabla_b g^{-T}\nabla_a g^T & \nabla_a f & \nabla_a g \\ \nabla_{ba}\mathcal{L} - \nabla_{bb}\mathcal{L}\nabla_b g^{-T}\nabla_a g^T & \nabla_b f & \nabla_b g \\ \nabla_a f^T - \nabla_b f^T \nabla_b g^{-T}\nabla_a g^T & & \end{bmatrix} \begin{pmatrix} d_a \\ d_{\lambda_f} \\ d_{\lambda_g} \end{pmatrix} = - \begin{pmatrix} L_\mu \\ 0 \\ f \end{pmatrix} \tag{21}$$

$$L_\mu = \nabla_a \varphi - \nabla_b g^{-1}\nabla_b \varphi + (\nabla_a f - \nabla_a g \nabla_b g^{-1}\nabla_b f)\lambda_f + A^{-1}\mu e \tag{22}$$

Pivoting (21) on the last element of the second block row gives us Equation (23), where $\nabla_a b$ has been used to simplify the expression. We use this expression to eliminate the last block column of the matrix in Equation (21) and give us Equation (24), where W is given by Equation (25).

$$d_{\lambda_g} = -\nabla_b g^{-1}(\nabla_{ba}\mathcal{L} + \nabla_{bb}\mathcal{L}\nabla_a b^T)d_a + \nabla_b g^{-1}\nabla_b f d_{\lambda_f} \tag{23}$$

$$\begin{bmatrix} W + A^{-1}Z & \nabla_a f + \nabla_a b \nabla_b f \\ \nabla_a f^T + \nabla_b f^T \nabla_a b^T & \end{bmatrix} \begin{pmatrix} d_a \\ d_{\lambda_f} \end{pmatrix} = - \begin{pmatrix} L_\mu \\ f \end{pmatrix} \tag{24}$$

$$W = \nabla_{aa}\mathcal{L} + \left(\nabla_{ab}\mathcal{L}\nabla_a b^T + \nabla_a b \nabla_{ba}\mathcal{L}\right) + \nabla_a b \nabla_{bb}\mathcal{L}\nabla_a b^T \tag{25}$$

We need to show that the system in Equation (24) has the same matrix and right-hand side as the system in Equation (14). With Equation (8), the lower left and upper right blocks in the matrices are equal, as is the upper block in the right-hand side. Expanding the individual Hessian terms in the upper left block of the matrix in (24), this block becomes

$$\begin{aligned} &\nabla^2_{aa}\varphi + \sum_i \lambda_{f_i}\nabla^2_{aa} f_i + \sum_j \lambda_{g_j}\nabla^2_{bb} g_j \\ &+ \left(\nabla^2_{ab}\varphi + \sum_i \lambda_{f_i}\nabla^2_{ab} f_i + \sum_j \lambda_{g_j}\nabla^2_{ab} g_j\right)\nabla_a b^T + \nabla_a b \left(\nabla^2_{ba}\varphi + \sum_i \lambda_{f_i}\nabla^2_{ba} f_i + \sum_j \lambda_{g_j}\nabla^2_{ba} g_j\right) \\ &+ \nabla_a b \left(\nabla^2_{bb}\varphi + \sum_i \lambda_{f_i}\nabla^2_{bb} f_i + \sum_j \lambda_{g_j}\nabla^2_{bb} g_j\right)\nabla_a b^T \\ &+ A^{-1}Z \end{aligned} \tag{26}$$

Because of Equations (13) and (19), this block is equal to $\nabla^2_{aa}\bar{\mathcal{L}} + A^{-1}Z$ when λ_f in the full space formulation (15) equals $\bar{\lambda}$ in the implicit function formulation (14) and functions are evaluated at the same primal variables a and b, which we have assumed. This expression is then equal to the upper left block in Equation (14). Because they have the same matrices and right hand sides, the systems (14) and (15) calculate the same search direction under the assumption of zero primal and dual infeasibility in external variables and constraints as long as the implicit function KKT matrix in (14) is nonsingular.

This implies that the implicit function formulation takes the same steps through an algorithm that uses this KKT matrix as an algorithm with the full space formulation that eliminates primal and dual infeasibility in coordinates of (5c) and b after every Newton step. The latter could be implemented within an existing nonlinear optimization solver as a structured restoration step, but doing so would require intrusive modification of the solver and familiarity with the solver's internal data structures. By contrast, the implicit function formulation may be implemented in any modeling framework that provides sufficient access to derivatives and interfaced with a wide variety of optimization solvers.

4 Case studies

To demonstrate the viability and potential advantages of the implicit function formulation, we apply it to dynamic optimization problems relevant to chemical engineering. This section presents and discusses these results.

4.1 Optimal control of a distillation column

Optimal control of a binary distillation column is used as an example of the dynamic optimization capabilities of Pyomo.DAE (Nicholson et al. (2018)). We solve a simple instance of this problem in which reflux ratio is manipulated to drive distillate concentration to a desired steady state. The distillation column model assumes vapor and liquid flow rates are constant among its 32 trays, does not consider an energy balance, and assumes a constant relative volatility α, and places its feed at tray 17. Differential equations are material balances on each tray and algebraic equations are vapor-liquid equilibrium equations. The formulation of the optimization problem is given by Equation (27). Here, L_t is the liquid flow rate in the rectification section, u_t is the reflux ratio, D is the constant distillate flow rate, S_t is the liquid flow rate in the stripping section, F is the constant feed flow rate, $x_{n,t}$ is the liquid-phase mole fraction at tray n and time t, $y_{n,t}$ is the vapor-phase mole fraction, and $\dot{x}_{n,t}$ is the time derivative of liquid-phase mole fraction. M_n is the holdup at tray n, while $\gamma = 1,000$ and $\rho = 1$ are objective weights.

$$
\begin{aligned}
\min \quad & \sum_t \left(\gamma(x_{1,t} - x_1^*)^2 + \rho(u_t - u^*)^2\right) \\
\text{s.t.} \quad & L_t = u_t D && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& V_t + L_t D && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& S_t = F + L_t && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& y_{n,t} = \frac{\alpha x_{n,t}}{1 + (\alpha - 1)\, x_{n,t}} && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& && \text{for all } n \text{ in } \{1, \ldots, 32\} \\
& \dot{x}_{1,t} = \frac{1}{M_1} V_t (y_{2,1} - x_{1,t}) && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& \dot{x}_{n,t} = \frac{1}{M_n} \left(L_t(x_{n-1,t} - x_{n,t}) - V_t(y_{n,t} - y_{n+1,t})\right) && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& && \text{for all } n \text{ in } \{2, \ldots, 16\} \\
& \dot{x}_{17,t} = \frac{1}{M_{17}} \left(F x_f + L_t x_{16,t} - S_t x_{17,t} - V_t(y_{17,t} - y_{18,t})\right) && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& \dot{x}_{n,t} = \frac{1}{M_n} \left(S_t(x_{n-1,t} - x_{n,t}) - V_t(y_{n,t} - y_{n+1,t})\right) && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& && \text{for all } n \text{ in } \{18, \ldots, 31\} \\
& \dot{x}_{32,t} = \frac{1}{M_{32}} \left(S_t x_{31,t} - (F - D_t) x_{32,t} - V_t y_{32,t}\right) && \text{for all } t \text{ in } \{t_0, \ldots, t_f\} \\
& \dot{x}_{n,t_i} = \frac{1}{\Delta t} \left(x_{n,t_i} - x_{n,t_{i-1}}\right) && \text{for all } t_i \text{ in } \{t_1, \ldots, t_f\} \\
& && \text{for all } n \text{ in } \{1, \ldots, 32\} \\
& x_{n,t_0} = \bar{x}_{n,t_0} && \text{for all } n \text{ in } \{1, \ldots, 32\} \\
& 1 \le u_t \le 5 && \text{for all } t \text{ in } \{t_0, \ldots, t_f\}
\end{aligned}
\tag{27}
$$

The code for a full-space implementation of the problem we solve can be found in the `examples/dae/` directory of the Pyomo repository, `https://github.com/pyomo/pyomo`.

Table 1: Characteristics of the optimization problem with full space and implicit function formulations

Formulation	# Var.	# Con.	Jac. nonzeros	Hess. nonzeros	# Iter.	Solve time
Full space	5,119	5,068	19,554	4,832	14	0.49 s
Implicit	3,282	3,232	108,832	107,250	13	10.5 s

Table 1 shows problem and solve statistics for the full space and implicit function formulations. Despite having fewer variables and constraints, the implicit function formulation has many more nonzeros in both the constraint Jacobian and Lagrangian Hessian matrices. This is due to a lack of sparsity in these matrices induced by the matrix factorization in Equation (7). Despite the presence of a matrix inverse, it may be beneficial to exploit and preserve sparsity in Equations (8) and (12). We defer this improvement to future work. In Table 1, "# Iter." refers to the number of iterations in the outer optimization problem. For this problem, the full space formulation converges in 0.49 CPU s, while the implicit function formulation converges in 10.5 CPU s. We also note that the full space and implicit function formulations lead to the same solution. The average relative error between variable values in the two solutions is 2.8×10^{-6}.

Table 2: Percentage of solve time spent in each activity for full and reduced space formulations

Formulation	I/O	IPOPT	Function eval.	Ext. solve	Jacobian	Hessian	other
Full space	62%	28%	5%	–	–	–	5%
Implicit	2%	7%	–	12%	23%	31%	25%

Table 2 shows the percentage of solve time spent in several different portions of the optimization solve for the full space and implicit function formulations. Here, the "I/O" category refers to time spent writing `.nl` files and reading `.sol` files in Python. "IPOPT" refers to time spend in the IPOPT library or executable for the optimization solve, not including function evaluations. We assume that this time is dominated by factorization of the KKT matrix. "Function eval." refers to time spent in function evaluations, including Jacobian and Hessian evaluation for the full space formulation. In the implicit function formulation, the solve of the external system $g(a, b) = 0$, Jacobian evaluation via Equation (8), and Hessian evaluation via Equation (12) are measured separately. In the full space formulation, we assume "other" is dominated by subprocess start-up and shut-down in the call to the IPOPT executable. In the implicit function formulation, "other" is dominated by time spent initializing the data structures necessary for derivative calculations and to communicate with CyIpopt.

For this small problem, the full space formulation spends most of its time writing `.nl` files, and only 28% actually in the IPOPT executable. Only 5% of the solve time is spent on function evaluations. In the implicit function formulation, by contrast, function and derivative evaluations combine for 66% of the solve time. Despite the massive increase in the number of matrix nonzeros shown in Table 1, the implicit function formulation only spends 7% of its time in the IPOPT algorithm, including matrix factorization and backsolve. In addition, the 25% of the solve time categorized as "other" indicates that our current implementation has significant overhead in data structure initialization for this small problem. We note that function and derivative evaluations, which are the computational bottleneck

in the implicit function formulation of this problem, are performed independently for the implicit function at each time discretization point. In this problem, the time domain is discretized into 52 points, so 52 processors could be used to speed up this portion of the solve. With perfect speedup, this would allow the optimization problem to solve in only 3.7 CPU s.

4.2 Chemical looping steady state simulation

A chemical looping combustion (CLC) reactor is a gas-solid hydrocarbon reactor in which fuel and air reactions occur in separate chambers, with a metal-oxide oxygen carrier looping between the two. Here we present results for the simulation of a reduction reactor involving methane and iron oxide operating at steady state in a counter-current moving bed configuration. The reduction reaction is shown in Equation (28).

$$CH_4 + 12Fe_2O_3 \rightarrow 2H_2O + CO_2 + 8Fe_3O_4 \tag{28}$$

The model equations form a DAE in which the continuous domain is distance along the length of the reactor. Differential variables are material and enthalpy flow rates and pressure. Algebraic equations and variables describe thermodynamics, hydrodynamics, transfer correlations, and the reaction rate. The full model equations for a dynamic (PDAE) instance of this model are provided in Appendix B. The steady state model is obtained by fixing time derivatives to zero. While the differential equations (50)-(53) only contain sums of bilinearities, the algebraic equations (54)-(97) are highly nonlinear and difficult to converge.

In the current simulation case study, the model has no inputs; inlet conditions are fixed. Further details about the model are provided by Ostace et al. (2018) and Okoli et al. (2020). The unit model and property packages necessary to construct this model are available on the IDAES repository at `https://github.com/idaes/idaes-pse` (Lee et al. (2021)). The simulation problem is a boundary value problem, and in discretized form amounts to solving a system of nonlinear equations. Variables are initialized to their inlet values, but this is not necessarily a good initial guess for the purpose of a nonlinear equation solver. Solving this boundary value problem is often performed as an initialization step before steady state or dynamic optimization.

To compare robustness and solve times of the full and reduced space formulations, we perform a parameter sweep that varies the gas and solid inlet temperatures. We attempt to solve the simulation problem in the full space and implicit function formulations for every combination of ten different gas and solid temperatures for a total of 100 problem instances. These instances have identical initialization routines, scaling factors, and solver options. In all of these instances, the length domain is discretized into 11 equally spaced discretization points. Figure 1 shows the convergence status of each combination of parameters for the full space and implicit function formulations. Here, an unsuccessful simulation may be due to timeout, iteration limit, function evaluation error, or convergence to an infeasible point.

Of the 100 problem instances, the full space formulation solves 42, while the implicit function formulation solves 82. The convergence results suggest that the simulation is more difficult to converge when gas and solid inlet temperatures are significantly different. The implicit function formulation solves many more instances than the full space formulation. In particular, it solves every instance for which inlet gas temperature is higher than inlet solid temperature.

Of the 42 instances solved by both formulations, the average solve time for the full space formulation is 0.17 s, while the average solve time for the implicit function formulation is

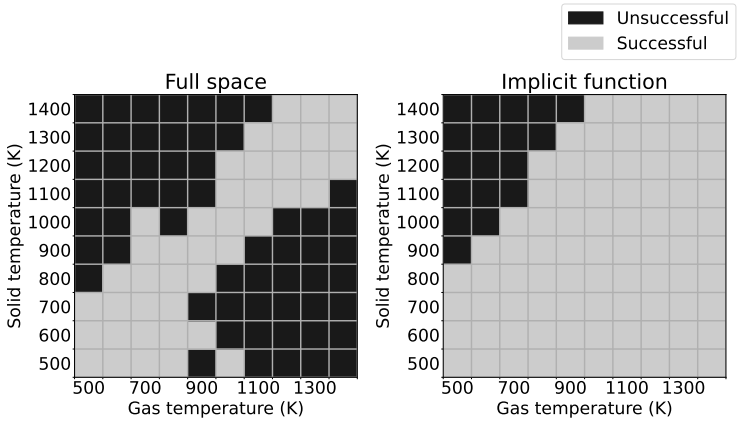

Figure 1: Convergence status of each combination of solid inlet temperature and gas inlet temperature for the full space and implicit function formulations.

3.9 s. Problem statistics for the single instance with a gas inlet temperature of 1,000 K and a solid inlet temperature of 1,200 K is given in Table 3, and a breakdown of time spent in different parts of this solve is given in Table 4. In this problem, the implicit function formulation has fewer equations and variables, fewer nonzeros in the Jacobian, and more nonzeros in the Hessian. Despite fewer nonzeros in the Jacobian, the implicit function formulation has a higher ratio of Jacobian nonzeros to variables and equations. This, and the larger number of nonzeros in the Hessian, could again be mitigated by fully exploiting potential sparsity in these derivatives, which we have deferred. In the full space formulation, the largest portion of solve time is again spent writing the `.nl` file, while the implicit function formulation spends 86% of the time evaluating the implicit function and its derivatives. As the external solve, Jacobian calculation, and Hessian calculation are performed independently at each spatial discretization point, these portions of the solve could be performed in parallel with up to 11 processors. With perfect speedup, a parallel implementation would improve the solve time of this instance by a factor of 4.5 for a solve time of 0.64 CPU s.

Table 3: Characteristics of the steady state CLC simulation problems with gas inlet temperature of 1,000 K and solid inlet temperature of 1,200 K

Formulation	# Var.	# Con.	Jac. nonzeros	Hess. nonzeros	# Iter.	Solve time
Full space	883	883	2,807	1,102	8	0.12 s
Implicit	180	180	1,801	1,630	10	2.9 s

Table 4: Percentage of solve times spent in each activity during the CLC steady state simulations with gas inlet temperature of 1,000 K and solid inlet temperature of 1,200 K

Formulation	I/O	Ipopt	Function eval.	Ext. solve	Jacobian	Hessian	other
Full space	63%	31%	6%	–	–	–	$< 1\%$
Implicit	6%	1%	–	64%	8%	14%	7%

4.3 Chemical looping dynamic optimization

We now present results for the dynamic optimization of the chemical looping combustion reactor described in Section 4.2. In our dynamic optimization problem, manipulated inputs are gas and solid inlet flow rates and are constrained to be piecewise constant on 120 s intervals. The control horizon contains 10 such intervals, each of which is discretized with two evenly spaced discretization points via an implicit Euler discretization. The objective function penalizes deviation between differential variables and inputs and their steady state set-point values at every point in time. The set-point values are computed by solving a steady state optimization problem with a target solid inlet flow rate and solid conversion. The model is a fully discretized partial differential and algebraic equation system in one spatial dimension and time. For the purpose of the implicit function formulation, differential and discretization variables and equations with respect to the length domain are considered algebraic and are solved by the implicit functions. The fully discretized full space model has 20,031 variables and 20,011 constraints, while the model seen by the optimizer in the implicit function formulation has 3,240 variables and 3,220 equations. All variables are initialized to their values at the initial steady state.

To compare solver robustness with the two formulations, we perform two parameter sweeps on the steady state setpoint used in the dynamic optimization problem. We calculate setpoints from ten target solid inlet flow rates and ten target conversions for a total of 100 different dynamic optimization problems in each parameter sweep. The first sweep solves an equality-constrained optimization problem, while the second includes an upper bound on gas phase inlet flow rate of 200 mol/s. All problem instances have identical initialization, scaling factors, and IPOPT options. A statement of the bound-constrained optimization problem is given by Equation (29). Here, z indexes the length domain of the reactor, GC is the set of gas phase components, and SC is the set of solid phase components. H_g and H_s and the gas and solid phase enthalpy holdups, and M_i and M_j are the gas and solid phase material holdups for components i and j. F_g is gas flow rate and F_s is solid flow rate. The gas inlet to the reactor is at z_0, while the solid inlet is at z_f.

$$
\begin{aligned}
\min \quad & \textstyle\sum_t \sum_z \left(\left(H_{g,t,z} - H^*_{g,z}\right)^2 + \sum_{j \in SC} \left(M_{j,t,z} - M^*_{j,z}\right)^2 \right) \\
& \textstyle + \sum_t \sum_z \left(\left(H_{g,t,z} - H^*_{s,z}\right)^2 + \sum_{i \in GC} \left(M_{i,t,z} - M^*_{i,z}\right)^2 \right) \\
\text{subject to} \quad & \text{Model equations (50) - (97)} \\
& \text{Initial conditions at steady state} \\
& \text{Boundary conditions} \\
& \text{Piecewise constant control inputs } F_g \text{ and } F_s \\
& \left.\begin{array}{l} 0 \le F_{g,t,z_0} \le 200 \text{ mol/s} \\ 0 \le F_{s,t,z_f} \end{array}\right\} \text{ for all } t \text{ in } \{t_0, \ldots, t_f\}
\end{aligned}
\tag{29}
$$

Convergence status for each equality-constrained dynamic optimization instance in the two formulations are shown in Figure 2. In this experiment, the full space formulation solves 60 out of 100 instances, while the implicit function formulation solves 89 out of 100 instances. For instances that converge in both formulations, the full space formulation takes an average of 53 s, while the implicit function formulation takes an average of 50 s. In these cases, the two formulations converge to the same solution. Table 5 gives problem statistics for an instance of this problem with a setpoint of 95% conversion and a solid flow rate of 700 kg/s, and Table 6 gives a breakdown of the solve times with this instance. While the full space and implicit function formulations take similar amounts of time to solve this instance, the

implicit function formulation converges in only six iterations as opposed to 15. The time per iteration is still significantly longer in the implicit function formulation than in the full space, although only by a factor of 2.3 for this instance, compared with a factor of approximately 20 in the previous small case studies. In this moderate-size optimization problem, the full space formulation spends most of its time in the IPOPT algorithm, including the linear system solve. The implicit function formulation of this problem again features a large increase in the number of Jacobian and Hessian nonzeros in the implicit function formulation, but only spends 3% of its solve time in IPOPT or the embedded linear system solve. In this solve, 90% of the time is spend in the implicit function evaluation and derivative calculations, which can be performed in parallel for each of the 20 time discretization points. If 20 processors were used and achieved perfect speedup of this portion of the solve, the implicit function formulation would converge in only 6.3 seconds, a significant improvement over the full space formulation.

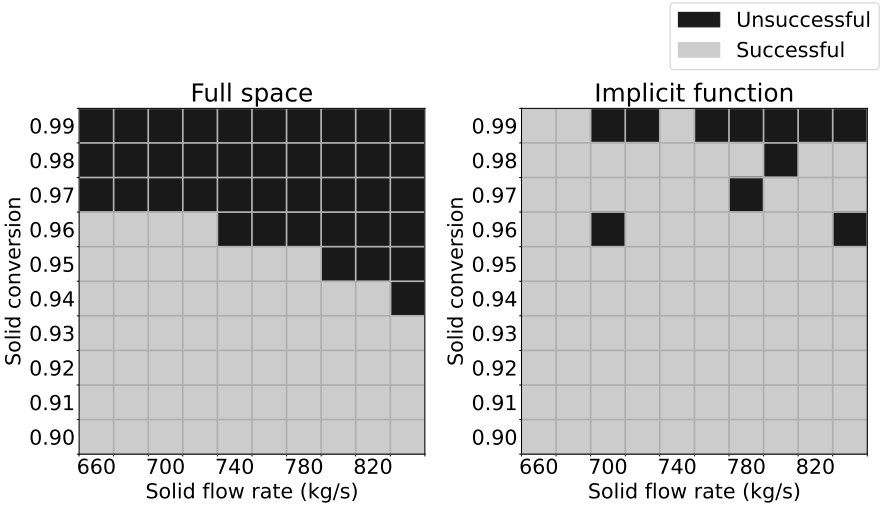

Figure 2: Convergence status for different setpoints, parameterized by solid inlet flow rate and target conversion, for the full space and implicit function formulations of the equality-constrained CLC dynamic optimization problem.

Table 5: Characteristics of an equality-constrained dynamic CLC optimization problem with a 95 % conversion and 700 kg/s setpoint

Formulation	# Var.	# Con.	Jac. nonzeros	Hess. nonzeros	# Iter.	Solve time
Full space	20,031	20,011	65,234	26,292	15	46.5 s
Implicit	3,240	3,220	263,960	264,060	6	43.2 s

In the eleven instances that do not converge with the implicit function formulation, the failure is due to an inability to solve a quadratic equation (the Ergun equation) for gas phase superficial velocity. At the particular values of the input variables sent to the

Table 6: Percentage of solve times spent in each activity during an equality-constrained CLC dynamic optimization solve with a 95 % and 700 kg/s setpoint

Formulation	I/O	Ipopt	Function eval.	Implicit solve	Jacobian	Hessian	other
Full space	4 %	90 %	6 %	–	–	–	$< 1\%$
Implicit	5%	3%	–	45%	15%	30 %	2 %

implicit function, the quadratic equation has no real solution for this variable, the Newton solve does not converge, and the outer optimization solve fails. This is analogous to a function evaluation error that could happen in the full space formulation if the function interface tries to evaluate, for instance, the square root of a negative number. Whereas we can reformulate or approximate a square root to avoid evaluation errors, we cannot easily reformulate a general nonlinear system to ensure it has at least one real solution. In IPOPT's AMPL interface, which is used by the full space formulation, these errors are handled by a reduction in step size. Reliability of the implicit function formulation could be further improved by triggering the same error handling procedure, which we leave for future work.

Convergence status for each bound-constrained dynamic optimization instance in the two formulations are shown in Figure 3. Here, the full space formulation converges 69 out of 100 instances in an average of 83 s, while the implicit function formulation converges all 100 instances in an average of 64 s. The bounds on gas inlet flow rate appear to be sufficient to prevent the primal variables from searching an area where superficial velocity cannot be determined by the Ergun equation. Adding this bound improves convergence in the full-space formulation as well, but the reason for this improvement is not easy to discern. Statistics for a single instance of this problem are given in Table 7 and a breakdown of the solve times is given in Table 8.

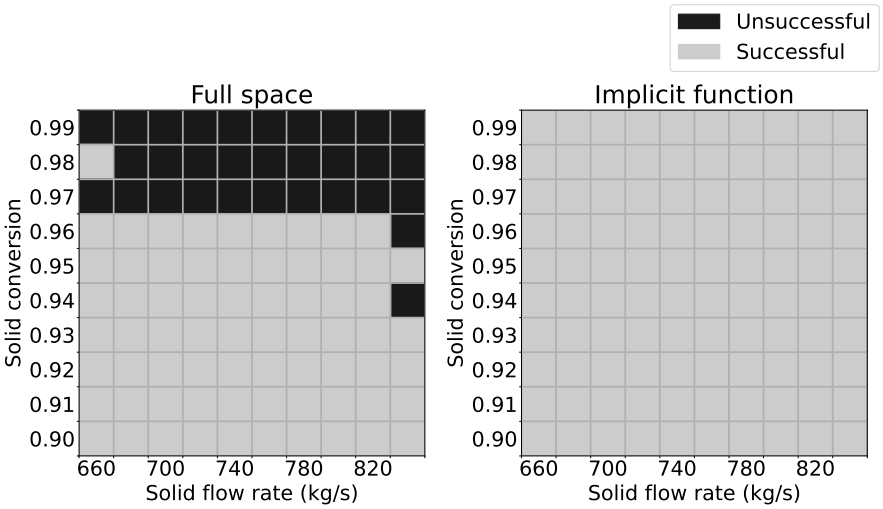

Figure 3: Convergence status with different setpoints for the full space and implicit function formulations of the bound-constrained CLC dynamic optimization problem.

Table 7: Characteristics of the bound-constrained dynamic CLC optimization problem with a 95 % and 700 kg/s setpoint

Formulation	# Var.	# Con.	Jac. nonzeros	Hess. nonzeros	# Iter.	Solve time
Full space	20,031	20,011	65,234	26,292	19	60 s
Implicit	3,240	3,220	263,960	264,060	8	64 s

These results show a significant improvement in robustness when using the implicit function formulation, which solves 48% more instances of the equality-constrained problem and 45% more instances of the bound-constrained problem.

5 Conclusion

We have implemented an implicit function formulation for the optimization of discretized index-1 DAE systems via nonlinear programming. The code for our implementation may be accessed in `external_pyomo_model.py` in the `pyomo/contrib/pynumero/interfaces` directory of the Pyomo repository, `https://github.com/pyomo/pyomo`, as of the 6.1 release of Pyomo. The application of our formulation to the optimal control of a distillation column and steady state simulation of a CLC reactor indicates that it is slower for small problems due to more expensive function and derivative evaluations. However, the application to CLC reactor simulation and CLC reactor dynamic optimization indicate that it has the potential to converge much more reliably, especially for challenging DAE optimization problems. In addition, for the moderate-sized CLC dynamic optimization problems, the solve time with the implicit function formulation is approximately the same as the full space formulation, indicating that increased function evaluation effort may be offset by less expensive factorization of the KKT matrix. We remark that in all case studies, both the full space and implicit function formulations would be able to solve all of the instances presented with better initialization and scaling, although good initial values and scaling factors are not easy to determine in a systematic way. In this regard, the implicit function formulation has an advantage that the optimization problem contains fewer variables and constraints that need to be initialized and scaled.

We hypothesize that the improved convergence reliability of the implicit function approach is because the implicit function Jacobians $\nabla_a \bar{f}$ and $\nabla_a b$, of Equations (7) and (8), are less likely to be poorly conditioned at points where the algebraic equations have been converged than at arbitrary infeasible points where the algebraic equations may have large residuals. While the embedded implicit function evaluation may also fail, we observe that this happens less frequently than failure of the optimization algorithm in the full space formulation.

There are many extensions to the current work that can be explored. We believe that

Table 8: Percentage of solve times spent in each activity during the bound-constrained CLC dynamic optimization solve with a 95 % and 700 kg/s setpoint

Formulation	I/O	Ipopt	Function eval.	Implicit solve	Jacobian	Hessian	other
Full space	3%	94%	3%	–	–	–	$< 1\%$
Implicit	4%	3%	–	46%	15%	30 %	2 %

the dynamic optimization of many different reactor and separator column models would benefit from this approach. In addition, the computational performance can be improved by exploiting sparsity in the implicit function derivatives, evaluating different implicit functions in parallel, and evaluating implicit functions exclusively in compiled code rather than in Python. This approach may be applied to implicit functions other than those naturally occurring in index-1 DAEs, and the formulation may be solved with optimization algorithms other than IPOPT for which a direct interface is available.

Our implicit function approach, which sits between the two extremes of fully sequential and fully simultaneous dynamic optimization, offers a welcome balance of convergence reliability and computational cost for large-scale DAE optimization problems.

6 Acknowledgements

This work was conducted as part of the Institute for the Design of Advanced Energy Systems (IDAES) with funding from the Simulation-Based Engineering, Crosscutting Research Program within the U.S. Department of Energy's Office of Fossil Energy

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-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. SAND2022-5132 O

References

Bock, H. & Plitt, K. (1984), 'A multiple shooting algorithm for direct solution of optimal control problems', *IFAC Proceedings Volumes* **17**(2), 1603–1608. 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984.

Bongartz, D. & Mitsos, A. (2017), 'Deterministic global optimization of process flowsheets in a reduced space using McCormick relaxations', *Journal of Global Optimization* **69**, 761–796.

Bynum, M. L., Hackebeil, G. A., Hart, W. E., Laird, C. D., Nicholson, B. L., Siirola, J. D., Watson, J.-P. & Woodruff, D. L. (2021), *Pyomo–optimization modeling in Python*, Vol. 67, third edn, Springer Science & Business Media.

Cuthrell, J. E. & Biegler, L. T. (1987), 'On the optimization of differential-algebraic process systems', *AIChE Journal* **33**(8), 1257–1270.

Duff, I. S. & Reid, J. K. (1978), 'An Implementation of Tarjan's Algorithm for the Block Triangularization of a Matrix', *ACM Trans. Math. Softw.* **4**(2), 137–147.

Eckstein, J. & Bertsekas, D. P. (1992), 'On the Douglas-Rachford splitting method and proximal point algorithm for maximal monotone operators', *Mathematical Programming* **55**, 293–318.

Gay, D. M. (1997), Hooking your solver to AMPL, Technical report, Bell Laboratories.

Goh, C. & Teo, K. (1988), 'Control parametrization: A unified approach to optimal control problems with general constraints', *Automatica* **24**(1), 3–18.

Hannemann, R. & Marquardt, W. (2007), 'Fast computation of the hessian of the lagrangian in shooting algorithms for dynamic optimization', *IFAC Proceedings Volumes* **40**(5), 105–110. 8th IFAC Symposium on Dynamics and Control of Process Systems.

Hart, W. E., Watson, J.-P. & Woodruff, D. L. (2011), 'Pyomo: modeling and solving mathematical programs in Python', *Mathematical Programming Computation* **3**(3), 219–260.

Hong, W., Wang, S., Li, P., Wozny, G. & Biegler, L. T. (2006), 'A quasi-sequential approach to large-scale dynamic optimization problems', *AIChE Journal* **52**(1), 255–268.

Kisala, T., Trevino-Lozano, R., Boston, J., Britt, H. & Evans, L. (1987), 'Sequential modular and simultaneous modular strategies for process flowsheet optimization', *Computers & Chemical Engineering* **11**(6), 567–579.

Kouzoupis, D., Quirynen, R., Frasch, J. & Diehl, M. (2015), 'Block condensing for fast nonlinear mpc with the dual newton strategy', *IFAC-PapersOnLine* **48**(23), 26–31. 5th IFAC Conference on Nonlinear Model Predictive Control NMPC 2015.

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. (2021), 'The IDAES process modeling framework and model library—Flexibility for process simulation and optimization', *Journal of Advanced Manufacturing and Processing* **3**(3), e10095.
URL: *https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/amp2.10095*

Mitsos, A., Chachuat, B. & Barton, P. I. (2009), 'McCormick-based relaxations of algorithms', *SIAM Journal on Optimization* **20**(2), 573–601.

Nicholson, B., Siirola, J. D., Watson, J.-P., Zavala, V. M. & Biegler, L. T. (2018), '`pyomo.dae`: A modeling and automatic discretization framework for optimization with differential and algebraic equations', *Mathematical Programming Computation* **10**(2), 187–223.

Okoli, C. O., Ostace, A., Nadgouda, S., Lee, A., Tong, A., Burgard, A. P., Bhattacharyya, D. & Miller, D. C. (2020), 'A framework for the optimization of chemical looping combustion processes', *Powder Technology* pp. 149–162.

Ostace, A., Lee, A., Okoli, C. O., Burgard, A. P., Miller, D. C. & Bhattacharyya, D. (2018), Mathematical modeling of a moving-bed reactor for chemical looping combustion of methane, *in* 'Proc. 13th International Symposium on Process Systems Engineering (PSE 2018)', Computer-Aided Chemical Engineering, Elsevier, San Diego, United States, pp. 325–330.

Ozyurt, D. B. & Barton, P. I. (2005), 'Cheap second order directional derivatives of stiff ODE embedded functionals', *SIAM Journal on Scientific Computing* **26**(5), 1725–1743.

Rodriguez, J., Parker, R., Laird, C., Nicholson, B., Siirola, J. & Bynum, M. (2021), 'Scalable Parallel Nonlinear Optimization with PyNumero and Parapint'. Preprint at http://www.optimization-online.org/DB_HTML/2021/09/8596.html.

Stuber, M. D., Scott, J. K. & Barton, P. I. (2015), 'Convex and concave relaxations of implicit functions', *Optimization Methods Software* **30**(3), 424–460.

Vassiliadis, V. S., Canto, E. B. & Banga, J. R. (1999), 'Second-order sensitivities of general dynamic systems with application to optimal control problems', *Chemical Engineering Science* **54**(17), 3851–3860.

Wächter, A. & Biegler, L. T. (2006), 'On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming', *Math. Program.* pp. 25–27.

Wilhelm, M. E., Le, A. V. & Stuber, M. D. (2019), 'Global optimization of stiff dynamical systems', *AIChE Journal* **65**(12), e16836.

Yoshio, N. & Biegler, L. T. (2021), 'A nested schur decomposition approach for multiperiod optimization of chemical processes', *Computers & Chemical Engineering* **155**, 107509.

A Hessian of the Lagrangian for the implicit function formulation

In this appendix we derive the form of the Hessian of the Lagrangian presented in Equation (12) from differentiation of (11). The gradient of the Lagrangian is

$$\nabla_a \bar{\mathcal{L}} = \nabla_a \varphi + \nabla_a b \nabla_b \varphi + \sum_i \bar{\lambda}_i \left(\nabla_a f_i + \nabla_a b \nabla_b f_i \right) + z \tag{30}$$

By differentiating, the Hessian of the Lagrangian is

$$\begin{aligned} \nabla^2_{aa} \bar{\mathcal{L}} =& \nabla^2_{aa} \varphi + \nabla_a b \nabla^2_{ba} \varphi + \nabla^2_{ab} \varphi \nabla_a b^T + \nabla_a b \nabla^2_{bb} \varphi \nabla_a b^T + \sum_j \nabla_b \varphi_j \nabla^2_{aa} b_j \\ &+ \sum_i \bar{\lambda}_i \left(\nabla^2_{aa} f_i + \nabla_a b \nabla^2_{ba} f_i + \nabla^2_{ab} f_i \nabla_a b^T + \nabla_a b \nabla^2_{bb} f_i \nabla_a b^T + \sum_j \nabla_b f_{ij} \nabla^2_{aa} b_j \right) \end{aligned} \tag{31}$$

where $\nabla_b f_{ij}$ is the $(i,j)^{\text{th}}$ (column-major) coordinate of the Jacobian matrix, corresponding to the derivative of the i^{th} function coordinate with respect to the j^{th} variable coordinate, and $\nabla^2_{aa} b_j$ is the $n_a \times n_a$ matrix that is the Hessian of the j^{th} coordinate of b. The Hessian

$\nabla^2_{aa} b$ can be obtained by twice differentiating g as in Equation (32), which uses a particular index k along the first rank of a third-order tensor. The equation holds for all k in $[1, \ldots, n_b]$.

$$0 = \nabla^2_{aa} g_k + \nabla_a b \nabla^2_{ba} g_k + \nabla^2_{ab} g_k \nabla_a b^T + \nabla_a b \nabla^2_{bb} g_k \nabla_a b^T + \left[\nabla_b g \nabla^2_{aa} b\right]_k \text{ for all } k \text{ in } [1, \ldots, n_b] \tag{32}$$

The bracketed term, $\left[\nabla_b g \nabla^2_{aa} b\right]$, is the product of a matrix and a $n_b \times n_a \times n_a$ third-order tensor. We evaluate by noting that its k^{th} coordinate along the first rank must correspond to a symmetric $n_a \times n_a$ matrix. Using slice notation to denote the matrix or vector obtained by fixing one or two particular indices of the third-order tensor, the term may be evaluated as shown in Equation (33), where the product on the right-hand side is a standard matrix-vector product.

$$\left[\nabla_b g \nabla^2_{aa} b\right]_{:ij} = \nabla_b g^T \nabla^2_{aa} b_{:ij} \tag{33}$$

Let H be the third-order tensor defined by (34).

$$H_{k::} = \nabla^2_{aa} g_k + \nabla_a b \nabla^2_{ba} g_k + \nabla^2_{ab} g_k \nabla_a b^T + \nabla_a b \nabla^2_{bb} g_k \nabla_a b^T \tag{34}$$

Then Equation (32) may be rewritten in terms of coordinates i and j along the second and third ranks of the tensor as shown in Equation (35).

$$0 = H_{:ij} + \nabla_b g^T \nabla^2_{aa} b_{:ij} \text{ for all } i, j \text{ in } [1, \ldots n_a,] \times [1, \ldots n_a,] \tag{35}$$

By the nonsingularity of $\nabla_b g$, we have

$$\nabla^2_{aa} b_{:ij} = -\nabla_b g^{-T} H_{:ij} \tag{36}$$

Which can be rewritten in terms of a coordinate along the first rank of $\nabla^2_{aa} b$,

$$\nabla^2_{aa} b_l = \sum_k - \left(\nabla_b g^{-T}\right)_{lk} H_{k::} = \sum_k - \left(\nabla_b g^{-T}\right)_{lk} \left(\nabla^2_{aa} g_k + \nabla_a b \nabla^2_{ba} g_k + \nabla^2_{ab} g_k \nabla_a b^T\right) \tag{37}$$

We now return to the Hessian of the Lagrangian, which from (31) we can rewrite as (38).

$$\begin{aligned} \nabla^2_{aa} \bar{\mathcal{L}} =& \nabla^2_{aa} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{aa} f_i \\ &+ \nabla_a b \nabla^2_{ba} \varphi + \sum_i \bar{\lambda}_i \nabla_a b \nabla^2_{ba} f_i + \nabla^2_{ab} \varphi \nabla_a b^T + \sum_i \bar{\lambda}_i \nabla^2_{ab} f_i \nabla_a b^T \\ &+ \nabla_a b \nabla^2_{bb} \varphi \nabla_a b^T + \sum_i \bar{\lambda}_i \nabla_a b \nabla^2_{bb} f_i \nabla_a b^T \\ &+ \sum_j \nabla_b \varphi_j \nabla^2_{aa} b_j + \sum_i \bar{\lambda}_i \sum_j \nabla_b f_{ij} \nabla^2_{aa} b_j \end{aligned} \tag{38}$$

We will reformulate the last two terms of the right-hand side of (38), which we define T_φ and T_f.

$$T_\varphi = \sum_j \nabla_b \varphi_j \nabla^2_{aa} b_j \tag{39}$$

$$T_f = \sum_i \bar{\lambda}_i \sum_j \nabla_b f_{ij} \nabla^2_{aa} b_j \tag{40}$$

Using the characterization of $\nabla^2_{aa} b$ given in Equation (37), with H_k as shorthand for $H_{k::}$,

$$T_\varphi = \sum_j \nabla_b \varphi_j \sum_k \left(-\nabla_b g^{-T}\right)_{kj} H_k \tag{41}$$

$$T_f = \sum_i \bar{\lambda}_i \sum_j \nabla_b f_{ij} \sum_k \left(-\nabla_b g^{-T}\right)_{kj} H_k \tag{42}$$

We now rearrange summations and products involving scalars,

$$T_\varphi = \sum_k H_k \sum_j \left(-\nabla_b g^{-T}\right)_{kj} \nabla_b \varphi_j \tag{43}$$

$$T_f = \sum_k H_k \sum_j \left(-\nabla_b g^{-T}\right)_{kj} \sum_i \nabla_b f_{ij} \bar{\lambda}_i \tag{44}$$

and combine these terms as they appear in (38):

$$T_\varphi + T_f = \sum_k H_k \sum_j \left(-\nabla_b g^{-T}\right)_{kj} \left(\nabla_b \varphi_j + \sum_i \nabla_b f_{ij} \bar{\lambda}_i\right) \tag{45}$$

This term may be rewritten with standard matrix-vector products as shown in Equation (46).

$$T_\varphi + T_f = \sum_k H_k \left[-\nabla_b g^{-1} \left(\nabla_b \varphi + \nabla_b f \bar{\lambda}\right)\right]_k \tag{46}$$

We define the factor in brackets as λ_g,

$$\lambda_g = -\nabla_b g^{-1} \left(\nabla_b \varphi + \nabla_b f \bar{\lambda}\right) \tag{47}$$

Then term $T_\varphi + T_f$ may be written as in Equation (48), where H_k has been expanded.

$$T_\varphi + T_f = \sum_k \left(\nabla^2_{aa} g_k + \nabla_a b^T \nabla^2_{ba} g_k + \nabla^2_{ab} g_k \nabla_a b + \nabla_a b^T \nabla^2_{bb} g_k \nabla_a b\right) \lambda_{g_k} \tag{48}$$

Moving the sums over i and k inward and combining like terms, the Hessian of the Lagrangian $\nabla^2_{aa} \bar{\mathcal{L}}$ may be written

$$\begin{aligned}
\nabla^2_{aa} \bar{\mathcal{L}} =& \nabla^2_{aa} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{aa} f_i + \sum_k \lambda_{g_k} \nabla^2_{aa} g_k \\
&+ \nabla_a b \left(\nabla^2_{ba} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{ba} f_i + \sum_k \lambda_{g_k} \nabla^2_{ba} g_k\right) \\
&+ \left(\nabla^2_{ab} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{ab} f_i + \sum_k \lambda_{g_k} \nabla^2_{ba} g_k\right) \nabla_a b^T \\
&+ \nabla_a b \left(\nabla^2_{bb} \varphi + \sum_i \bar{\lambda}_i \nabla^2_{bb} f_i + \sum_k \lambda g_k \nabla^2_{bb} g_k\right) \nabla_a b^T
\end{aligned} \tag{49}$$

Which is the desired form presented in Equation (12).

B Equations for the chemical looping PDAE model

This section presents the PDAE equations of the undiscretized chemical looping combustion reduction reactor model. Sets of gas and solid components are given in Table 9, variables and parameters used in the PDAE model are given in Table 10, and differential and algebraic equations are given in Table 11. All variables presented are functions of time and the reactor's length domain and equations are repeated at every point in time and space.

Symbol	Description	Elements
GC	Set of components in the gas phase	{CH_4, CO_2, H_2O}
SC	Set of components in the solid phase	{Fe_2O_3, Fe_3O_4, Al_2O_3}

Table 9: Sets used in the description of the PDAE model

Symbol		Name	Units
M_i	$i \in GC$	Gas component holdup	mol/m
H_g		Gas energy holdup	kJ/m
M_j	$j \in SC$	Solid component holdup	kg/m
H_s		Solid energy holdup	kJ/m
f_i	$i \in GC$	Gas component flow rate	mol/s
$f_{H,g}$		Gas enthalpy flow rate	kJ/s
f_j	$j \in SC$	Solid component flow rate	kg/s
$f_{H,g}$		Solid enthalpy flow rate	kJ/s
F_g		Gas total molar flow rate	mol/s
F_s		Solid total mass flow rate	kg/s
v_g		Gas superficial velocity	m/s
v_s		Solid superficial velocity	m/s
y_i	$i \in GC$	Gas component mole fraction	—
T_g		Gas phase temperature	K
P		Gas pressure	bar
x_i	$j \in SC$	Solid component mass fraction	—
T_s		Solid phase temperature	K
ρ_g		Gas phase molar density	mol/m^3
$\rho_{\text{mass},g}$		Gas phase mass density	kg/m^3
$\rho_{g,i}$	$i \in GC$	Gas component density	mol/m^3
μ		Gas phase viscosity	kg/m/s
μ_i	$i \in GC$	Gas component viscosity	kg/m/s
$\hat{H}_g$		Gas molar enthalpy	kJ/mol
$\hat{H}_{g,i}$	$i \in GC$	Gas component molar enthalpy	kJ/mol
k_i	$i \in GC$	Gas component thermal conductivity	kJ/m/K/s
k		Gas phase thermal conductivity	kJ/m/K/s
MW_g		Gas average molecular weight	kg/mol
$\rho_{s,p}$		Solid particle density	kg/m^3
$\rho_{s,s}$		Solid skeletal density	kg/m^3

ϵ_p		Particle porosity	—
$\hat{H}_s$		Solid specific enthalpy	kJ/kg
$\hat{H}_{s,j}$	$j \in SC$	Solid component specific enthalpy	kJ/kg
Cp_s		Solid specific heat capacity	kJ/kg/K
$\mathrm{Cp}_{s,j}$	$j \in SC$	Solid component molar heat capacity	kJ/mol/K
N_i	$i \in GC$	Rate of generation in the gas phase	mol/m/s
N_j	$j \in SC$	Rate of generation in the solid phase	mol/m/s
α_i	$i \in GC$	Gas component stoichiometric coefficient	—
α_j	$j \in SC$	Solid component stoichiometric coefficient	—
ξ		Reaction rate extent	mol/m/s
r_{rxn}		Reaction rate	mol/m/s
k_{rxn}		Reaction rate coefficient	$(\mathrm{mol/m^3})^{-0.3}\mathrm{m/s}$
X		Oxygen carrier conversion	—
Q_g		Gas phase heat transfer rate	kJ/m/s
Q_s		Solid phase heat transfer rate	kJ/m/s
Re_p		Particle Reynolds number	—
Pr		Prandtl number	—
Nu_p		Particle Nusselt number	—
h_{gs}		Gas-solid heat transfer coefficient	$\mathrm{kJ/m^2/K/s}$
MW_i	$i \in GC$	Gas component molecular weight	kg/mol
MW_j	$j \in SC$	Solid component molecular weight	kg/mol
R		Gas constant	kJ/mol/K
a_p		Particle solid volume fraction	—
E_A		Activation energy	kJ/mol
d_p		Particle diameter	m
$\rho_{j,s}$	$j \in SC$	Solid pure component skeletal density	$\mathrm{kg/m^3}$
ΔH_{rxn}		Enthalpy of reaction	kJ/mol
l		Reactor length	m
A		Reactor cross-sectional area	$\mathrm{m^2}$
A_g		Reactor gas phase cross-sectional area	$\mathrm{m^2}$
A_s		Reactor solid phase cross-sectional area	$\mathrm{m^2}$
ϵ		Void fraction of reactor	—

Table 10: Variables and parameters participating in the PDAE model

Equation		Name	
$l\frac{\partial M_i}{\partial t} = \frac{\partial f_i}{\partial x} + lN_i$	$i \in GC$	Gas material balance	(50)
$l\frac{\partial H_g}{\partial t} = \frac{\partial f_{H,g}}{\partial x} + lQ_g$		Gas energy balance	(51)
$l\frac{\partial M_j}{\partial t} = \frac{\partial f_j}{\partial x} + lN_j \mathrm{MW}_j$	$j \in SC$	Solid material balance	(52)
$l\frac{\partial H_s}{\partial t} = \frac{\partial f_{H,s}}{\partial x} + lQ_s + \Delta H_{rxn}\xi$		Solid energy balance	(53)
$M_i = y_i \rho_i A_g$	$i \in GC$	Gas material holdup equation	(54)
$H_g = A_g \rho_g \hat{H}_g$		Gas energy holdup equation	(55)
$M_j = x_j \rho_{s,p} A_s$	$i \in SC$	Solid material holdup equation	(56)
$H_s = A_s \rho_{s,p} \hat{H}_s$		Solid energy holdup equation	(57)
$f_i = y_i F_g$	$i \in GC$	Gas component flow equation	(58)
$f_{H,g} = \hat{H}_g F_g$		Gas enthalpy flow equation	(59)
$f_j = x_j F_g$	$j \in SC$	Solid component flow equation	(60)
$f_{H,s} = \hat{H}_s F_s$		Solid enthalpy flow equation	(61)
$1 = \sum_i y_i$	$i \in GC$	Gas component fraction sum	(62)
$1 = \sum_j x_j$	$j \in SC$	Solid component fraction sum	(63)
$v_g = \frac{F_g}{A_g \rho_g}$		Gas velocity equation	(64)
$v_s = \frac{F_s}{A_s \rho_{s,p}}$		Solid velocity equation	(65)
$\frac{\partial P}{\partial x} = 150\frac{(1-\epsilon)^2 \mu (v_g + v_s)}{d_p^2 \epsilon^3} + \frac{7}{4}\frac{\rho_{g,\mathrm{mass}}(1-\epsilon)(v_g+v_s)^2}{d_p \epsilon^3}$		Ergun equation	(66)
$A_g = (1-\epsilon)\, A$		Gas area equation	(67)
$A_s = \epsilon A$		Solid area equation	(68)
$\rho_i = \rho_g y_i$	$i \in GC$	Gas component density equation	(69)
$P = \rho_g R T_g$		Ideal gas equation	(70)

$\mu_i = \frac{A_{\mu,i} T_g^{B_{\mu,i}}}{1 + C_{\mu,i}/T_g + D_{mu,i}/T_g^2}$	$i \in GC$	Component viscosity equation	(71)
$\mu = \sum_{i \in GC} \frac{y_i \mu_i}{\sum_{i' \in GC} y_{i'} \left(\frac{MW_{i'}}{MW_i}\right)^{1/2}}$		Dynamic viscosity equation	(72)
$\hat{H}_i = A_{H,i} T_g + B_{H,i} T_g^2 + C_{H,i} T_g^3 + D_{H,i} T_g^4 + \frac{E_{H,i}}{T} + F_{H,i}$	$i \in GC$	Gas component enthalpy equation	(73)
$H_g = \sum_{i \in GC} y_i \hat{H}_i$		Gas molar enthalpy equation	(74)
$k_i = \frac{A_{k,i} T_g^{B_{k,i}}}{1 + C_{k,i}/T_g + D_{k,i}/T_g^2}$	$i \in GC$	Gas component conductivity	(75)
$k_g = \sum_{i \in GC} \frac{y_i k_i}{\sum_{i'} y_{i'} A_{ii'}^{1/2}}$		Gas thermal conductivity equation	(76)
$A_{ii'} = \frac{\left(1 + \left(\frac{k_{i'}}{k_i}\right)^{1/2} \left(\frac{MW_{i'}}{MW_i}\right)^{1/4}\right)^2}{\left(8\left(1 + \frac{MW_{i'}}{MW_i}\right)\right)^{1/2}}$	$i, i' \in GC$	Gas conductivity binary coefficient equation	(77)
$\rho_{g,\text{mass}} = \text{MW}_g \rho_g$		Gas mass density equation	(78)
$\text{MW}_g = \sum_{i \in GC} y_i \text{MW}_i$		Gas molecular weight equation	(79)
$\hat{H}_j = A_{H,j} T_s + B_{H,j} T_s^2 + C_{H,j} T_s^3 + D_{H,j} T_s^4 + \frac{E_{H,j}}{T_s} + F_{H,j}$	$j \in SC$	Solid component enthalpy equation	(80)
$\hat{H}_s = \sum_{j \in SC} x_j \hat{H}_j$		Solid specific enthalpy equation	(81)
$\text{Cp}_j = A_{H,j} + B_{H,j} T_s + C_{H,j} T_s^2 + D_{H,j} T_s^3 + \frac{E_{H,j}}{T^2}$	$j \in SC$	Solid component heat capacity equation	(82)
$\text{Cp}_s = \sum_{j \in SC} \frac{1}{\text{MW}_j} x_j \text{Cp}_j$		Solid specific heat capacity equation	(83)
$\rho_{s,p} = (1 - \epsilon_p) \rho_{s,s}$		Solid particle density equation	(84)
$\rho_{s,s} = \left(\sum_{j \in SC} \frac{x_j}{\rho_{j,s}}\right)^{-1}$		Solid skeletal density equation	(85)
$N_i = \alpha_i r_{\text{rxn}} A_s$	$i \in GC$	Gas component generation equation	(86)
$N_j = \alpha_j \xi$	$j \in SC$	Solid component generation equation	(87)

$\xi = r_{\text{rxn}} A_s$	Reaction extent equation	(88)
$r_{\text{rxn}} = \frac{3 x_{Fe_2O_3} \rho_{s,s} a_{vol} k_{\text{rxn}} \rho_{CH_4}^{1.3}}{\text{MW}_{Fe_2O_3} \rho_{mol,s} r_g} (1 - X)^{2/3}$	Reaction rate equation	(89)
$k_{\text{rxn}} = k_0 \exp\left(\frac{E_A}{R T_s}\right)$	Rate coefficient equation	(90)
$X = \frac{x_{Fe_3O_4}}{x_{Fe_3O_4} - (\text{MW}_{Fe_3O_4} \frac{\alpha_{Fe_3O_4}}{\alpha_{Fe_2O_3}}) x_{Fe_2O_3}}$	Oxygen carrier conversion equation	(91)
$Q_g d_p = -6 h_{gs} A_s \left(T_g - T_s\right)$	Gas heat transfer equation	(92)
$Q_s d_p = 6 h_{gs} A_s \left(T_g - T_s\right)$	Solid heat transfer equation	(93)
$h_{gs} d_p = \text{Nu}_p k_g$	Heat transfer coefficient equation	(94)
$\text{Nu}_p^3 = \left(2 + 1.1 \text{Re}_p^{1.8}\right) \text{Pr}$	Nusselt number equation	(95)
$\text{Pr} = \frac{\text{Cp}_s \mu}{k_g}$	Prandtl number equation	(96)
$\text{Re}_p = \frac{v_g d_p \rho_{g,\text{mass}}}{\mu}$	Reynolds number equation	(97)

Table 11: Equations used by the PDAE model