A FRAMEWORK FOR MODELING AND OPTIMIZING DYNAMIC SYSTEMS UNDER UNCERTAINTY

Bethany Nicholson*[1] and John Siirola[1]

[1]Center for Computing Research, Sandia National Laboratories, Albuquerque, NM 87185

Abstract

Algebraic modeling languages (AMLs) have drastically simplified the implementation of algebraic optimization problems. However, there are still many classes of optimization problems that are not easily represented in most AMLs. These classes of problems are typically reformulated before implementation, which requires significant effort and time from the modeler and obscures the original problem structure or context. In this work we demonstrate how the Pyomo AML can be used to represent complex optimization problems using high-level modeling constructs. We focus on the operation of dynamic systems under uncertainty and demonstrate the combination of Pyomo extensions for dynamic optimization and stochastic programming of a semibatch reactor model under abnormal conditions.

Keywords

Stochastic Programming, Dynamic Optimization, Optimal Control, Parameter Estimation

Introduction

A common challenge rarely mentioned or addressed in the optimization research community is the difficulty associated with implementing cutting-edge methods or industrial-sized optimization problems. This includes not only the implementation of new solution techniques for a particular class of optimization problems but also the implementation of models that span several classes of optimization problems. By a "class" of optimization problems we mean the labels researchers apply to their work based on the high-level modeling constructs they consider; e.g., stochastic programming, bilevel optimization, or dynamic optimization. The challenge in most cases is that general optimization solvers are not designed to handle the high-level modeling constructs that we use to concisely represent and understand complex optimization problems. Constructs such as multiple objectives, disjunctions, uncertainty scenario trees, and differential equations are common components found in models in the literature and are easy to write down on paper but their implementations in code can often be rather complicated. These constructs must be reformulated before they can be expressed in most AMLs and therein lies the problem. Often these reformulations are tedious or non-trivial to implement and, given an implementation of a previously reformulated problem, it can be difficult to reverse engineer the intent or goal of the original problem. Furthermore, applying nested reformulations to problems that include several high-level modeling constructs can require a considerable amount of time and coding effort. Finally, there are usually several ways to reformulate a high-level construct. By implementing a reformulation rather than the high-level construct itself it becomes difficult to experiment with alternative reformulations. The work in this paper aims to show how many of these challenges can be overcome by modeling these types of problems with high-level modeling constructs and offering libraries of generalized reformulations that can be applied automatically. The examples in this paper consider problems that contain modeling constructs from both dynamic optimization and stochastic programming. However, we want to emphasize that the work presented here can be extended to other classes of optimization problems and their combinations. In the next section we intro-

*To whom all correspondence should be addressed
blnicho@sandia.gov

duce the key components of the software package used in this work. We then discuss a novel optimization modeling paradigm that enables the rapid and easy implementation of complex structured optimization problems. Finally, we demonstrate our technique using two illustrative examples of optimizing dynamic systems under uncertainty.

Pyomo

Pyomo (Hart et al., 2011) is a free and open-source algebraic modeling language (AML) developed in Python. It provides users extreme flexibility in expressing and manipulating optimization models along with access to other Python packages for model analysis and plotting. In addition to standard AML features, Pyomo is well-suited for meta-algorithm development and generic implementations of high-level modeling constructs and transformations. As evidence of that, Pyomo includes extensions for stochastic programming, generalized disjunctive programming, bi-level problems, and differential algebraic equation(DAE) constrained optimization. These extensions can be used together to quickly and easily formulate very sophisticated models which would be difficult to represent using other modeling packages. In this work we focus on two of the high-level Pyomo extensions, `pyomo.dae` for dynamic optimization and `PySP` for stochastic programming. We provide brief overviews of these extensions in the next two sections.

Pyomo.dae

DAE constrained optimization problems allow modelers to incorporate detailed dynamic models directly in an optimization problem. However, solving these problems can be challenging as general optimization solvers do not have the ability to represent differential equations. Therefore, dynamic optimization problems need to be transformed into some other form before they can be directly solved by an optimization algorithm. There are several approaches for transforming the problem, but a common theme in many AMLs is that the transformations must be applied manually, which can be tedious and error prone. The `pyomo.dae` extension allows modelers to formulate dynamic optimization problems in their natural form, directly writing differential equations as differential equations. The package then provides a library of automated transformations from which the modeler can select to convert the DAE model into a “solvable” form. The transformations included in the package apply a simultaneous discretization to the dynamic model and return a nonlinear programming problem back to the user. The package allows users to experiment with different discretization schemes and resolutions without modifying the underlying dynamic model. `pyomo.dae` is still under active development and additional features and transformations will be available in the near future. More information about the package can be found in Nicholson et al. (2016).

PySP

Stochastic programming is a useful technique for finding optimal (or near-optimal) decisions while directly accounting for uncertainty. However, this approach has not seen wide-spread use in the process industries primarily due to the difficulties of expressing the model and the size of the resulting optimization problem. `PySP`(Watson et al., 2011) is a Pyomo extension which addresses these challenges. It allows users to express stochastic programs by specifying a deterministic base model formulated in Pyomo and a scenario tree model defining the problem stages and uncertain parameters. `PySP` also provides two strategies for solving stochastic programs. The first approach builds the deterministic equivalent (extensive form) of the model, which then can be solved with standard deterministic optimization solvers. For stochastic programs where the individual scenario models are large or there are many scenarios, formulating and solving the extensive form quickly becomes computationally intractable, both in terms of memory and solution time. The second approach provided in `PySP` addresses this challenge by decomposing the extensive form into a series of smaller subproblems and employing an iterative approach to converge the original problem. `PySP` provides several approaches for automatically decomposing the problem, including both scenario-based (Progressive Hedging) and stage-based (Bender’s decomposition) decompositions. As `PySP` has explicit knowledge of the underlying stochastic program structure, it can directly exploit distributed computing platforms by both generating and solving the subproblems in parallel. More information on the package can be found in Watson et al. (2011).

High-level Modeling Constructs and Reformulations

High-level modeling constructs like those provided in the aforementioned Pyomo extensions allow users to formulate models in a much more natural form. This leads to fewer coding mistakes and more intuitive implementations of complex optimization problems. In addition, it separates the model specification from the reformulation or solution technique used to solve it. This defers decisions about how a model will be solved until solution time rather than during model implementation. This separation lets the user quickly experiment with different reformulations while reusing most (if not all) of their model code.

General implementations of common reformulations also drastically reduce the expert knowledge required to use certain techniques by shifting the burden of expert implementation from the user to the implementation developer. Similarly, generic reformulations also transfer the reformulation debugging responsibilities from the modeler to the transformation developer. Furthermore, treating each reformulation as a distinct transformation that can be applied to a model opens up the possibility of applying multiple reformulations in series and allows the user to experiment with different combinations of reformulations.

Outside of modeling extensions for specific classes of optimization, Pyomo also has core modeling constructs for expressing model structure. This allows modelers to further abstract their implementation into different submodels and link them together. This functionality has been extensively used in modeling power systems where submodels are written for each network component(e.g. buses, lines, generators) and then linked together to create a model of the entire power network(Greenhall et al., 2012; Friedman et al., 2013). Being able to represent underlying model structure in a straightforward way opens the door for rapid development of generalized solution algorithms that exploit particular structures. These structure capturing features are not explicitly used in the implementations of the examples in this work however they are fundamental for several Pyomo extensions including `PySP`.

Semibatch Reactor Model

We demonstrate the modeling concepts described in this paper using a dynamic model of the semibatch reactor from Abel and Marquardt (2000). This model considers the following series of highly exothermic reactions

$$A \longrightarrow B \longrightarrow C \tag{1}$$

We assume that component A is fed into a solvent-filled reactor and reacts to form the desired product B. Component B then partially reacts to form the undesired product C. This model considers a reactor vessel equipped with two heat-exchange systems, a reactor jacket and an internal coil. The dynamic model equations are shown below. They include component, mass, and heat balances.

$$\dot{C}_a = \frac{F_a}{V_r} - k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a \tag{2}$$

$$\dot{C}_b = k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a - k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b \tag{3}$$

$$\dot{C}_c = k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b \tag{4}$$

$$\dot{V}_r = \frac{F_a M_{W_a}}{\rho_r} \tag{5}$$

$$\begin{aligned}(\rho_r c_{p_r})\dot{T}_r = & \frac{F_a M_{W_a} c_{p_r}}{V_r}(T_f - T_r) \\ & - k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a \Delta H_1 \\ & - k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b \Delta H_2 \\ & + \alpha_{w,j}\frac{A_j}{V_{r,0}}(T_{w,j} - T_r) + \alpha_{w,c}\frac{A_c}{V_{r,0}}(T_{w,c} - T_r)\end{aligned} \tag{6}$$

Table 1. Parameters for semibatch reactor model

Parameter	Value	Parameter	Value
k_1	15.01 1/s	A_j	5.0 m^2
k_2	85.01 1/s	A_c	3.0 m^2
E_1	30,000.0 kJ/kmol	V_j	0.9 m^3
E_2	40,000.0 kJ/kmol	V_j	0.07 m^3
R	8.314 kJ/kmol/K	ρ_w	700.0 kg/m^3
M_{W_a}	50.0 kg/kmol	c_{p_w}	3.1 kJ/kg/K
ρ_r	1,000.0 kg/m^3	$C_a(t_0)$	0.0 kmol/m^3
c_{p_r}	3.9 kJ/kg/K	$C_b(t_0)$	0.0 kmol/m^3
T_f	300.0 K	$C_c(t_0)$	0.0 kmol/m^3
ΔH_1	-40,000.0 kJ/kmol	$T_r(t_0)$	300.0 K
ΔH_2	-50,000.0 kJ/kmol	$V_r(t_0) = V_{r,0}$	1.0 m^3
$\alpha_{w,j}$	0.8 kJ/s/m^2/K	$\alpha_{w,jfail}$	0.3 kJ/s/m^2/K
$\alpha_{w,c}$	0.7 kJ/s/m^2/K		

Values for the model parameters and initial conditions for the differential equations are given in Table 1. The control inputs for this model are the feed rate of component A, F_a, and the cooling medium temperatures in the jacket and in the coil, $T_{w,j}$ and $T_{w,c}$ respectively. For the sake of simplicity we assume that there is a single cooling medium source and the jacket and coil temperatures are equal, reducing the number of control inputs to two.

In the next two sections we use the semibatch reactor model in two illustrative examples of stochastic dynamic optimization problems: parameter estimation and optimal control. The point of these examples is not to promote novel model formulations or solution techniques but rather to promote novel implementations of complex optimization problems. While the main contribution of this work is a drastically simplified implementation, we omit the model code for the sake of brevity. The full implementations of these examples are available as part of the Pyomo project.

Stochastic Parameter Estimation

We first demonstrate how the combination of `PySP` and `pyomo.dae` can be used to formulate and solve dynamic parameter estimation problems. We use the dynamic semibatch reactor model described in the previous section to simulate the system behavior under different experimental conditions and then add Gaussian noise to the concentration and reactor temperature profiles to obtain simulated experimental measurements. We assume that measurements are taken every 36 minutes (the time step used by Abel and Marquardt (2000)). To vary the experimental conditions we introduce step changes in the control inputs F_a and $T_{w,c}$ three hours into the batch. We also assume that different sets of measurements were taken for each experiment, simulating the case where a sensor might fail or there are only enough experimental resources to measure a subset of the variables of interest. Table 2 notes the step changes present in each of three experiments and the measurements recorded for each experiment. We assume that we are trying to estimate the Arhenius equation parameters, $k_1, k_2, E_1,$ and E_2.

Table 2. Experimental conditions for parameter estimation with missing measurements

Experiment	Step Change In	Measurements
1	F_a	C_a, C_b, C_c, T_r
2	$T_{w,c}$	T_r
3	$F_a, T_{w,c}$	C_b, T_r

Our implementation of this problem first formulates a dynamic optimization model for each experiment. The code to generate this model is reused with slight modifications to the objective function depending on which measurements are available. The objective function minimizes the squared difference between our measurements and our model. The dynamic model is discretized using collocation over finite elements and we ensure that a finite element boundary lies at each measurement time. These individual dynamic models are then fed into `PySP` as separate scenarios within a two-stage stochastic programming problem. The Arhenius equation parameters, $k_1, k_2, E_1, \text{and} E_2$, constitute the first stage decision variables to force the optimization to identify a single estimate across all experiments, whereas the time series for each experiment form the scenario-specific second stage variables. We use `PySP` to build the scenario tree for this problem and generate the extensive form of the model. We then solve the extensive form of the problem using IPOPT(Wächter and Biegler, 2006).

The parameter estimates for two cases are given in Table 3. In the 'All Meas.' case we assume that measurements of C_a, C_b, C_c, and T_r are available for every experiment. In the 'Missing Meas.' case we assume that we only have a subset of measurements for each experiment according to Table 2. We see that we obtain parameters that are relatively close to the actual parameter values and, as expected, our estimates get worse in the case of missing measurements.

Table 3. Parameter estimates

	$k_1(1/s)$	$k_2(1/s)$	$E_1(kJ/kmol)$	$E_2(kJ/kmol)$
Actual	15.01	85.01	30,000	40,000
All Meas.	16.84	81.19	30,322	39,861
Missing Meas.	20.69	77.42	30,850	39,697

This example is meant to illustrate the ease with which a parameter estimation problem can be formulated and solved using the modeling constructs provided in Pyomo. We have demonstrated this on a parameter estimation problem using a dynamic model and experimental data with different operating conditions and measured variables. Our implementation could also be extended to address experimental data with different measurement frequencies or batch times.

Stochastic Optimal Control

In this example we recreate the optimal control problem developed by Abel and Marquardt (2000) with some minor modifications. The motivation of this problem is to find an optimal control strategy that maximizes the expected amount of product B while ensuring that reactor operating limits can be maintained in the case of partial cooling system failure. In other words, we want to find a nominal control strategy for the batch such that we can always find a feasible updated set of controls to save the batch in case of cooling failure.

This model explicitly considers the case where the cooling jacket around the reactor fails. Abel and Marquardt (2000) formulate this as a scenario-based optimal control problem by noting that the jacket could fail at any time during the batch run. The stochastic uncertainty in the problem is the time of failure. After cooling jacket failure, the temperature in the jacket is modeled using the following differential equation:

$$(\rho_w c_{p_w} V_j)\dot{T}_{w,j} = \alpha_{w,jfail} A_j \frac{V_r}{V_{r,0}} (T_{w,j} - T_r) \tag{7}$$

where a smaller value of the heat transfer coefficient $\alpha_{w,jfail}$ captures the the reduced cooling potential of the jacket. The value of this new parameter is given in Table 1.

In order to convert the continuous set of failure time realizations into something tractable for an optimization solver, we only consider a finite, discrete set of failure times over the batch run. In this example we consider 9 different equally-likely failure times uniformly spaced at 36 minute intervals over a 6 hour nominal batch time. One of the main differences between our example and the one in (Abel and Marquardt, 2000) is how we deal with the batch time after jacket failure. Abel and Marquardt (2000) identify the optimal amount of time to extend the batch run after jacket failure up to a maximum of 6 hours. For most of the failure scenarios they found that the optimal batch extension time was the maximum, 6 hours. In our work we assume that the batch extension time is always 6 hours, which simplifies the model implementation and objective function specification. This also results in failure scenarios that have different total batch run times.

Abel and Marquardt (2000) also include an additional constraint on the adiabatic end temperature of the reactor. They explain that this constraint limits the reactor temperature in case of a total cooling system failure, and represents a simpler way to address this extreme case than including it as an additional scenario. Equation 9 captures this additional constraint.

$$T_{ad} = T_r + \frac{V_r\left[(-\Delta H_1 - \Delta H_2)C_a - \Delta H_2 C_b\right]}{\rho_r c_{p_r} V_r + \rho_w c_{p_w}(V_j + V_c)} \tag{8}$$

Equation 10 shows the complete optimization problem corresponding to a single scenario, or realization, of

$$\begin{aligned}
&\min_{F_a, T_{w,c}} \; C_b(t_f) && (9)\\
&\text{s.t. Eqs. 2 to 6 and Eq. 9}\\
&\int_0^{t_f} F_a dt = 20 \text{ kmol}\\
&C_a(t_f) \le 0.5 \text{ kmol}/m^3\\
&T_{ad} \le 453 \text{ K}\\
&0 \le F_a \le 180.0 \text{ kmol/h}\\
&288 \text{ K} \le T_{w,c} \le 432 \text{ K}\\
&t_f = t_{fail} + 6 \text{ h}\\
&T_{w,j}(t) = \begin{cases} \text{Eq. 7} & t \le t_{fail} \\ \text{Eq. 8} & t > t_{fail} \end{cases}
\end{aligned}$$

the failure time, t_{fail}.

We include bounds on the control inputs and specifications for the total amount of component A fed to the reactor and the concentration of A at the end of the batch run.

Similar to the previous example, we solve this problem by formulating each scenario as a separate model, discretizing the dynamic equations using `pyomo.dae`, and creating the extensive form of the overall problem using `PySP`. Plots of the control inputs and concentration profiles for the scenario $t_{fail} = 3.0$h are shown in Figure 1. For the sake of brevity we don't include plots for the other scenarios. The final concentration of component B in each scenario is reported in Table 4. Our results are within 3% of those reported in Abel and Marquardt (2000), with the discrepancy primarily attributable to the difference in treatment of the batch completion time.

Table 4. Final product concentrations

t_{fail}	$C_b(t_f)$	t_{fail}	$C_b(t_f)$
0.6 h	9.35 kmol/m^3	3.6 h	9.81 kmol/m^3
1.2 h	9.50 kmol/m^3	4.2 h	9.85 kmol/m^3
1.8 h	9.64 kmol/m^3	4.8 h	9.71 kmol/m^3
2.4 h	9.73 kmol/m^3	5.4 h	9.01 kmol/m^3
3.0 h	9.78 kmol/m^3	(N/A)	9.07 kmol/m^3

Abel and Marquardt (2000) implement their solution strategy using a shooting approach that directly leverages several Fortran packages for numerical integration and optimization. In contrast, this work completely isolated the dynamic model from the solution methodology.

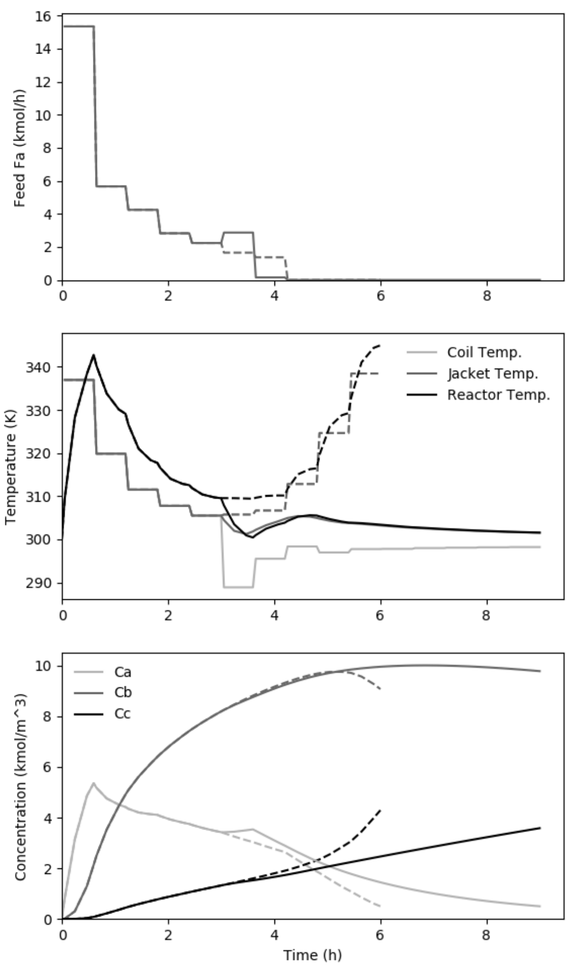

Figure 1. Operating conditions for the nominal case (dashed lines) and the case where $t_{fail} = 3$ h (solid lines). (Top) Feed flow rate for component A (Middle) Cooling coil and jacket temperatures and the reactor temperature (Bottom) Concentration profiles for each component

We were able to implement the model equations almost exactly as they appear in the paper and rely on Pyomo packages to manipulate the model. Furthermore, everything was formulated, solved, and plotted using a single software platform (Python). The entire implementation (including result plotting) requires less than 300 lines of code, divided among the deterministic dynamic model specification (60%), discretization (2%), stochastic problem formulation (18%), and result plotting (20%).

Conclusions

In this paper we have demonstrated the power and modeling flexibility of the modeling language Pyomo and some of its extensions. Pyomo is ideally suited for rapid model and algorithm development and capable of representing and solving a large range of optimization problems. We have illustrated these features by presenting two examples of combining the `pyomo.dae` and `PySP` packages for dynamic optimization and stochastic programming.

Acknowledgments

This work was conducted as part of the Institute for the Design of Advanced Energy Systems (IDAES) with funding from the Office of Fossil Energy, Cross-Cutting Research, U.S. Department of Energy. Sandia is a multi-mission laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000. SAND2016-11854 C

References

Abel, O. and Marquardt, W. (2000). Scenario-integrated modeling and optimization of dynamic systems. *AIChE Journal*, 46(4):803–823.

Friedman, Z., Ingalls, J., Siirola, J., and Watson, J.-P. (2013). Block-oriented modeling of superstructure optimization problems. *Computers & Chemical Engineering*, 57:10–23.

Greenhall, A., Christie, R., and Watson, J.-P. (2012). Minpower: A power systems optimization toolkit. In *2012 IEEE Power and Energy Society General Meeting*, pages 1–6. IEEE.

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

Nicholson, B., Siirola, J. D., Watson, J.-P., Zavala, V. M., and Biegler, L. T. (2016). `pyomo.dae`: A modeling and automatic discretization framework for optimization with differential and algebraic equations. *Mathematical Programming Computation*. Manuscript submitted for publication.

Wächter, A. and Biegler, T. L. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57.

Watson, J.-P., Woodruff, D., and Hart, W. (2011). Pysp: Modeling and solving stochastic programs in python. *Mathematical Programming Computation*, 3:219–260.