Optimal PID Controller Tuning Using Stochastic Programming Techniques

Jose A. Renteria, Yankai Cao, Alexander W. Dowling, and Victor M. Zavala
Dept. of Chemical and Biological Engineering, University of Wisconsin-Madison, 1415 Engineering Dr, Madison, WI 53706

DOI 10.1002/aic.16030
Published online November 27, 2017 in Wiley Online Library (wileyonlinelibrary.com)

We argue that stochastic programming provides a powerful framework to tune and analyze the performance limits of controllers. In particular, stochastic programming formulations can be used to identify controller settings that remain robust across diverse scenarios (disturbances, set-points, and modeling errors) observed in real-time operations. We also discuss how to use historical data and sampling techniques to construct operational scenarios and inference analysis techniques to provide statistical guarantees on limiting controller performance. Under the proposed framework, it is also possible to use risk metrics to handle extreme (rare) events and stochastic dominance concepts to conduct systematic benchmarking studies. We provide numerical studies to illustrate the concepts and to demonstrate that modern modeling and local/global optimization tools can tackle large-scale applications. The proposed work also opens the door to data-based controller tuning strategies that can be implemented in real-time operations. © 2017 American Institute of Chemical Engineers *AIChE J*, 64: 2997–3010, 2018

Keywords: PID control, stochastic programming, risk, uncertainty, robustness

Introduction

PID controller tuning is one of the classical problems of process control. Despite the fact that advanced control architectures such as model predictive control (MPC) are currently available,[1,2] the controller tuning problem remains industrially relevant because either advanced architectures require highly reliable low-level controllers to implement control policies or because the deployment of advanced architectures might not practical in certain settings (e.g., due to limited budgets or computing capabilities[3]). Tuning is a challenging task because one must ensure that the controller is capable of handling a wide range of operational events that include set-point changes, exogenous disturbances, physical constraints, and interactions with other controllers.[4] Optimization-based controller tuning provides a flexible and systematic framework to address these issues because it can naturally factor in loop interactions, different control configurations (e.g., cascades), as well as diverse sets of constraints (e.g., saturation) and objective functions (e.g., tracking and economic).

The literature on optimal PID controller tuning is extensive; consequently, we only provide representative work to highlight the major capabilities and limitations of state-of-the-art techniques. In the pioneering work of Astrom,[5,6] a nonconvex optimization framework is presented to tune single-loop PID controllers to perform disturbance rejection. Under that framework, constraints on Nyquist stability are imposed in the frequency domain to ensure robustness. A similar frequency-domain formulation is presented in Ref. 7. In Ref. 8, a linear-quadratic-regulator (LQR) optimal control approach (in the time domain) is used to tune PID controllers. A limitation of this work is that it can only handle linear models. In Ref. 9, a black-box optimization approach is presented to tune single-loop PID controllers based on H_∞ criteria (see Ref. 10 for a review). A limitation of black-box optimization is that it is not scalable. Analytical tuning techniques such as direct synthesis and internal model control (IMC) are goal-driven techniques that also inherit the high flexibility of optimization techniques.[11,12] In Ref. 13, a computational framework is presented to conduct controller tuning using a wide range of techniques (including optimization in both time and frequency domains). The framework also provides high flexibility to handle diverse controller configurations and performance functions.

A common limitation of the aforementioned techniques and tools is that they do not explicitly account for uncertainty in the design process (this is often handled after-the-fact through repetitive testing). As a result, real-time controller re-tuning (also known as auto-tuning or adaptive tuning[14]) is often necessary after controllers are designed and deployed.[15] In Ref. 16, a multi-model paradigm combined with LQR techniques is used to factor in model uncertainty directly in the tuning process. The authors show that the associated robust tuning problem can be cast as an optimization problem with linear matrix inequality constraints (LMIs). Limitations of this approach are that it can only handle linear models and that the associated semidefinite programming problems become quickly intractable. In Ref. 17, a PID tuning method for nonlinear dynamical systems is presented. Here, the nonlinearity is handled by creating a family of linear models that are deemed accurate in a certain parametric domain. It is shown that the tuning approach guarantees that the closed-loop system is stable with

Correspondence concerning this article should be addressed to V. M. Zavala at victor.zavala@wisc.edu

high probability for uniformly-distributed model uncertainty (in the form of stochastic matrices for the state-space representation). Moreover, the approach can be applied to multi-loop PID tuning. A limitation of this approach is that it is limited to LQR formulations and does not capture general uncertainties such as exogenous disturbances.

In this work we propose to use stochastic programming (SP) techniques to perform controller tuning tasks. SP inherits the flexibility of optimization-based tuning schemes but can also directly accommodate a wide range of uncertainties experienced in real-time operations. In addition, SP techniques can be used to exploit historical data to create uncertainty representations, can use diverse sampling techniques to generate average and extreme operational scenarios, and can use risk metrics to mitigate extreme events. We also demonstrate that modern optimization modeling and solution tools can handle challenging tuning formulations that involve large numbers of scenarios and complex models. Our developments focus on continuous-time formulations and simple architectures with coupled PID controllers. The concepts, however, can be applied in a wider context that include frequency-domain, complex architectures, and different control laws (e.g., hybrid[3,18]). The use of SP techniques has become increasingly popular in the context of MPC.[19–21] To the best of our knowledge, however, SP techniques have not been used in the context of controller tuning.

The article is structured as follows. We first motivate the discussion by providing a deterministic optimization-based tuning formulation. This setting is then expanded into a stochastic programming setting that captures uncertainties. We also discuss the use of historical data, sampling techniques, risk measures, and computational tools. We then present case studies, conclusions, and future work.

Controller Setting

We consider a dynamical system of the form:

$$\dot{x}(t)=f(x(t),u(t),d(t),p), \quad t\in\mathcal{T} \tag{0.1a}$$

$$y(t)=h(x(t),d(t),p), \quad t\in\mathcal{T} \tag{0.1b}$$

$$x(0)=x_0 \tag{0.1c}$$

where $t\in\mathcal{T}\subseteq\mathbb{R}$ is time, $x(t)\in\mathbb{R}^{n_x}$ are the states with initial conditions x_0, $y(t)\in\mathbb{R}^{n_y}$ are the outputs, $u(t)\in\mathbb{R}^{n_u}$ are the controls, $d(t)\in\mathbb{R}^{n_d}$ are the disturbances (e.g., sensor errors and exogenous forcings), and $p\in\mathbb{R}^{n_p}$ are model parameters. The state and output mappings $f:\mathbb{R}^{n_x}\times\mathbb{R}^{n_u}\times\mathbb{R}^{n_d}\times\mathbb{R}^{n_p}\rightarrow\mathbb{R}^{n_x}$ and $h:\mathbb{R}^{n_x}\times\mathbb{R}^{n_d}\times\mathbb{R}^{n_p}\rightarrow\mathbb{R}^{n_y}$ are assumed to be continuously differentiable. We define trajectories over the time horizon $\mathcal{T}$ for states as $x_\mathcal{T}$ and we use a similar notation for time-dependent controls $u_\mathcal{T}$, outputs $y_\mathcal{T}$, and disturbance $d_\mathcal{T}$ trajectories.

We assume a square closed-loop multivariable system with $n:=n_y=n_u$ and define PID loops of the form:

$$u_i(t)=K_P^i\epsilon_i(t)+K_I^i\int^t\epsilon_i(\tau)d\tau+K_D^i\dot{\epsilon}_i(t), \quad i=1,..,n \tag{0.2}$$

where $K_P^i,K_I^i,K_D^i\in\mathbb{R}_+$ are the design *settings* for controller in loop i. The closed-loop error is:

$$\epsilon_i(t)=\bar{y}_i(t)-y_i(t), \quad i=1,\ldots,n \tag{0.3}$$

where $\bar{y}(t)\in\mathbb{R}^n$ are output set-points and $\epsilon(t)\in\mathbb{R}^n$ is the output error. The controller equations can be written in vector form as:

$$u(t)=\mathrm{diag}(K_P)\epsilon(t)+\mathrm{diag}(K_I)I(t)+\mathrm{diag}(K_D)\dot{\epsilon}(t), \quad t\in\mathcal{T} \tag{0.4a}$$

$$\dot{I}(t)=\epsilon(t), \quad t\in\mathcal{T}. \tag{0.4b}$$

We define the *controller settings* vector $\pi=(K_P,K_I,K_D)\in\mathbb{R}_+^{n_\pi}$ with $n_\pi=3\cdot n$. In this work we restrict our attention to PID control laws (due to their wide applicability and ease in implementation). We highlight, however, that more complicated control laws can also be considered in the proposed framework.[18]

The controller specifications π must be tuned to perform tracking of the set-points and to reject disturbances. An approach to obtain controller specifications in a systematic manner is by solving an *optimal tuning problem* of the form:

$$\min_\pi \phi(x_\mathcal{T},y_\mathcal{T},\epsilon_\mathcal{T},u_\mathcal{T},\pi) \tag{0.5a}$$

$$\dot{x}(t)=f(x(t),u(t),d(t),p), \quad t\in\mathcal{T} \tag{0.5b}$$

$$y(t)=h(x(t),d(t),p), \quad t\in\mathcal{T} \tag{0.5c}$$

$$\epsilon(t)=\bar{y}(t)-y(t), \quad t\in\mathcal{T} \tag{0.5d}$$

$$u(t)=\mathrm{diag}(K_P)\epsilon(t)+\mathrm{diag}(K_I)I(t)+\mathrm{diag}(K_D)\dot{\epsilon}(t), \quad t\in\mathcal{T} \tag{0.5e}$$

$$\dot{I}(t)=\epsilon(t), \quad t\in\mathcal{T} \tag{0.5f}$$

$$0\leq g(x(t),u(t),y(t),\pi), \quad t\in\mathcal{T} \tag{0.5g}$$

$$x(0)=x_0. \tag{0.5h}$$

Here, $\phi(\cdot)$ is a scalar cost function and $g(\cdot)$ is a function vector that captures diverse constraints such as terminal constraints as well as bounds on outputs, states, controls (i.e., actuator saturation), and controller settings.

A classical cost function for the tuning problem is a deviation measure of the form[13]:

$$\phi(x_\mathcal{T},y_\mathcal{T},\epsilon_\mathcal{T},u_\mathcal{T},\pi)=\int_\mathcal{T}\|\epsilon(\tau)\|_\beta d\tau+\int_\mathcal{T}\|u(\tau)\|_\beta d\tau, \tag{0.6}$$

with $\beta\geq 1$ (typically $\beta\in\{1,2,\infty\}$). The cost function, however, can be any suitable function that captures other aspects of operations such as economics, safety, and environment.

The *data* of the tuning problem is defined by the random vector $\xi=(x_0,\bar{y}_\mathcal{T},d_\mathcal{T},p)$. This data vector contains initial states, set-points, disturbances (sensor errors and exogenous forcings), and model parameters.

We consider the compact form of the optimal tuning problem (0.5):

$$\min_\pi \ \varphi(\pi,\xi) \tag{0.7a}$$

$$\text{s.t. } \pi\in\Pi(\xi), \tag{0.7b}$$

where $\varphi(\cdot)$ is the cost function (expressed in compact form) and $\Pi(\xi)$ represents the feasible set imposed by the dynamical model and the constraints of the tuning problem (0.5b)–(0.5h). The compact form representation is used to simplify the notation. Problems (0.5) and (0.7) are equivalent from a computational stand-point.

Optimization-based tuning formulations can be used to enforce different types of constraints, to directly capture loop

coupling (by embedding the dynamic model in the tuning formulation), and to optimize diverse performance metrics. We note, however, that the tuning formulation can be computationally challenging because of the embedded dynamical system (which might be stiff and exhibit a high degree of coupling) and because the problem formulation is inherently nonconvex (even if the dynamics are linear). Conversely, an important feature of the controller tuning formulation is that it only has n_π degrees of freedom (the dimension of the vector of design specifications π). We will see that this enables the use of scalable solution methods. We also highlight that by eliminating the control law constraints (0.5e), one obtains a standard optimal control problem (as that solved in MPC in real-time). Consequently, one can think of optimal controller tuning as a restricted form of MPC (similar in spirit to other implementations such as affine-control MPC laws[22]).

Stochastic Programming

An important goal in controller tuning is to ensure that the controller settings lead to optimal (or close-to-optimal) performance under a wide range of operational events of interest. The tuning formulation (0.7), however, only uses a single event. To enable the design of highly reliable controllers, we formulate a stochastic programming (SP) version of the tuning problem. In this formulation, the problem *data are modeled as random variables* Ξ. We consider the following SP formulation:

$$\min_{\pi} \quad \mathbb{E}_{\Xi}[\varphi(\pi,\Xi)] \tag{0.8a}$$

$$\text{s.t.} \quad \pi \in \Pi(\Xi), \quad \text{a.s.} \tag{0.8b}$$

Here, $\mathbb{E}_{\Xi}[\cdot]$ is the expectation operator (with respect to random variable Ξ and with associated density $p_{\Xi}(\cdot)$) and (0.8b) enforces that the constraints are satisfied with probability one or almost surely (a.s.). In other words, the constraint $\pi \in \Pi(\xi)$ must hold for all possible realizations ξ of the random variables. We denote the optimal solution of the above problem as π^*. From a controlled tuning stand-point, data realizations are to be interpreted as *operational events* (characterized as combinations of set-points, disturbances, sensor errors, and model parameters) that the controller is exposed to. Clearly, there can exist an astronomical (in fact, infinite if the random variables are continuous) number of potential events. For instance, if we consider five independent and discrete random variables and ten possible scenarios per random variable, we have $10^5 = 100,000$ possible combinations.

Risk measures

Thinking about the tuning problem from a SP perspective reveals the fundamental conflicts that arise in tuning tasks. Conflicts arise because performance in certain operational events (realizations) might be affected as we seek to improve performance in other events. The essence of SP is to find a solution (e.g., controller settings) that resolve such conflicts by shaping the probability distribution of the cost $\varphi(\cdot)$ in a desired way. For instance, minimizing expected value performance (as is done in formulation (0.8)) will emphasize performance over most likely realizations while running the risk of poor performance in some less probable (but potentially damaging) scenarios. Minimizing a risk measure can ensure that a controller simultaneously manages average and extreme scenarios.

A popular risk measure used in SP is the conditional value at risk (CVaR). One can define CVaR for the cost function as $\text{CVaR}_{1-\alpha}(\varphi(\pi,\Xi)) = \min_{\nu} \mathbb{E}_{\Xi}\left[\nu + \alpha^{-1}[\varphi(\pi,\Xi)-\nu]_+\right]$. One can thus minimize CVaR by formulating a SP problem of the form:

$$\min_{\pi,\nu} \quad \mathbb{E}_{\Xi}\left[\nu + \alpha^{-1}[\varphi(\pi,\Xi)-\nu]_+\right] \tag{0.9a}$$

$$\text{s.t.} \quad \pi \in \Pi(\Xi), \quad \text{a.s.} \tag{0.9b}$$

Here, $\alpha \in (0,1]$ is a probability level, $\nu \in \mathbb{R}$ is an auxiliary variable, and $[\cdot]_+ = \max\{0,\cdot\}$ is the max function. The above formulation has the property that the optimal value of the auxiliary variable ν^* is the $(1-\alpha)$ quantile of the distribution of the optimal cost $\varphi(\pi^*,\Xi)$. This quantile is also known as the value-at-risk (VaR). Moreover, one can show that the optimal objective value of (0.9) (minimum value of CVaR) satisfies $\mathbb{E}_{\Xi}\left[\nu^* + \alpha^{-1}[\varphi(\pi^*,\Xi)-\nu^*]_+\right] = \mathbb{E}[\varphi(\pi^*,\Xi) \mid \varphi(\pi^*,\Xi) \geq \nu^*]$. In other words, CVaR is the expectation in the $1-\alpha$ tail of the distribution. From this perspective, one can clearly see that $\text{CVaR}_{1-\alpha}(\varphi(\pi,\Xi)) = \mathbb{E}[\varphi(\pi,\Xi)]$ when $\alpha = 1$ (the tail is the entire distribution) and $\text{CVaR}_{1-\alpha}(\varphi(\pi,\Xi)) = \max_{\xi}\varphi(\pi,\xi)$ when $\alpha \to 0$ (the tail is the worst-case instance). Consequently, CVaR is a generalized metric that spans expected value and worst-case (robust) settings.

We recall that a worst-case formulation can also be formulated without using CVaR as:

$$\min_{\pi,\eta} \quad \eta \tag{0.10a}$$

$$\text{s.t.} \quad \pi \in \Pi(\Xi), \quad \text{a.s.} \tag{0.10b}$$

$$\varphi(\pi,\Xi) \leq \eta, \text{ a.s.} \tag{0.10c}$$

A key property of any controller settings π^* that minimize CVaR is that they provide *stochastic dominance* guarantees.[23] This means that we can guarantee that *no other controller settings* π *exist* that can improve the cost distribution $\varphi(\pi^*,\Xi)$. Stochastic dominance can be defined mathematically as follows: we say that a cost distribution (which we define simply as $\Xi^* := \varphi(\pi^*,\Xi)$) dominates an alternate cost distribution (which we define simply as $\Xi := \varphi(\pi,\Xi)$) to first-order if and only if:

$$F_{\Xi^*}(\nu) \geq F_{\Xi}(\nu), \quad \forall \nu \in \mathbb{R}. \tag{0.11}$$

Moreover, we say that stochastic dominance holds to second-order if and only if:

$$\int_{-\infty}^{\nu} F_{\Xi^*}(\nu')d\nu' \geq \int_{-\infty}^{\nu} F_{\Xi}(\nu')d\nu', \quad \forall\, \nu \in \mathbb{R}. \tag{0.12}$$

Here, $F_{\Xi}(\nu) = \int_{-\infty}^{\nu} p_{\Xi}(\nu')d\nu'$ is the cumulative density function (CDF) of the random variable Ξ up to a given level v and $p_{\Xi}(\nu)$ is the density function of Ξ evaluated at v. Similarly, $F_{\Xi^*}(\nu)$ and $p_{\Xi^*}(\nu)$ are the cumulative and density functions of Ξ^*. Stochastic dominance of first-order is a strong condition that is in general difficult to satisfy in practice. This is because the condition implies that the distribution of Ξ^* has more area than the distribution of Ξ up to any threshold v (e.g., the distribution $p_{\Xi^*}(\cdot)$ is always shifted towards the left of the distribution $p_{\Xi}(\cdot)$ up to any threshold v). Stochastic dominance of second order is weaker in the sense that it only guarantees that the area of $p_{\Xi^*}(\cdot)$ is (on average) greater than that of $p_{\Xi}(\cdot)$ up to any threshold v (i.e., the distribution is on average to the left).

In summary, the key result of interest in our context is that, by finding the controller settings π^* that minimize CVaR, we can guarantee that the associated cost $\varphi(\pi^*, \Xi)$ satisfies stochastic dominance of second order with respect to any other alternate controller settings π with associated cost $\varphi(\pi, \Xi)$. This property is important and, notably, is not guaranteed to be satisfied by other popular risk measures such as the mean-variance (which is often used to shape the tails of the distribution). Another limitation of using the mean-variance as a risk measure is that minimizing this measure reduces the tail of the cost distribution from both sides (symmetrically), and thus affects instances with low and high cost (but in practice we are only interested in controlling instances with high cost). CVaR, conversely, focuses only the tail associated to high costs, thus giving better performance.[24]

Stochastic dominance provides a systematic approach to *benchmarking alternative controllers*. To illustrate this, we discuss how modern numerical optimization solvers are currently benchmarked. In a benchmarking exercise, a given optimization solver is tested in thousands of problem instances (available in libraries such as CUTEr) and the CDF on a given performance metric (such as solution time) is generated. One can thus say that the solver dominates a competing solver if its CDF is above that of the competing solver (i.e., stochastic dominance holds). This approach has become the standard approach to benchmarking optimization solvers.[25] This is because this framework is based on solid statistical foundations and is easy to interpret. In a controller setting, we can interpret operational scenarios as benchmark problem instances under which alternative controllers are tested. Consequently, we can compare controllers by comparing their cost CDFs.

Scenario generation and data sources

The cost function of the SP problem (0.9) involves an expectation operator that cannot be evaluated explicitly (i.e., it is a multi-dimensional integral). Moreover, when the random variables are continuous, one cannot guarantee that the constraints can be satisfied with probability one (the problem is infinite-dimensional in the probability space). Consequently, to solve the SP problem, one must resort to approximation techniques such as sample average approximation (SAA) and sparse grids. In both of these techniques, we approximate the SP problem (0.9) by using a discrete (finite) set of data realizations $\xi \in \Omega$ to obtain the finite-dimensional problem:

$$\min_{\pi,\nu} \quad \sum_{\xi\in\Omega} w(\xi)\left[\nu+\alpha^{-1}[\varphi(\pi,\Xi)-\nu]_+\right] \tag{0.13a}$$

$$\text{s.t.} \quad \pi \in \Pi(\xi),\ \xi \in \Omega. \tag{0.13b}$$

Here, $w(\xi) \in [0, 1]$ is the probability of realization $\xi \in \Omega$ and we denote the number of scenarios as $|\Omega|$ (the cardinality of the set).

Monte Carlo sampling is an approach to generate scenarios from the underlying probability distributions (this is done in a random and unbiased manner). Sparse grids is another popular technique that generates scenarios systematically (in a biased manner) to cover the space of the random variables. When probability distributions for the random variables are not available to generate samples, one can use *historical data* to create empirical distributions. In industrial settings this is beneficial because one normally has access to a wide range of historical events (e.g., for set-point changes and disturbances) that can be used as realizations. Moreover, system identification techniques can be used to build models from operational data and these estimation techniques can provide distributions for model parameters.

In the absence of historical data, a practitioner can also model random variables by using generic distributions such as Gaussian, Bernoulli, Weibull, and uniform distributions. Gaussian distributions are commonplace and can easily capture basic correlations while uniform distributions are practical because they can be used to define ranges of operation (it is assumed that each realization occurs with equal probability in the range). Expert knowledge can also be used to generate hypothetical operational events (and their associated probabilities) that the controller must be able to withstand.

By formulating the controller tuning problem as an SP, it becomes possible to use a rich machinery of computational and analysis techniques that can aid the design of highly robust controllers and that can help assess their performance limits. For instance, we have already observed that risk measures can be used to shape the distribution of the controller cost and to provide dominance guarantees. We highlight that the proposed work also opens the door to systematic data-based controller tuning strategies that are implemented in real-time operations. Under such strategies, the controllers are retuned as more operational scenarios are collected in real-time. In the following sections, we illustrate how to use Monte Carlo sampling and *inference analysis* techniques to determine when more scenario information can no longer improve the performance of the controller settings (the settings are optimal). In addition, we discuss how to use sparse grids to design controllers that can withstand worst-case scenarios.

SAA and inference analysis

From a real-time operations stand-point, it is important to realize that it is extremely difficult (if not impossible) to guarantee that the controllers will be able to withstand all possible realizations of set-points, disturbances, and model errors. In other words, even if a controller works in most operational situations, there will always be a small probability that the controller will face an event that has never been seen in operations. It is thus our objective to ensure that the controller works optimally with high probability.

In SP, one uses inference techniques to assess the optimality of a solution in a *statistical sense*.[26] Here, we aim to determine the probability (confidence) that a given set of controller specifications (obtained with a finite set of realizations) will be optimal. These techniques also provide insights on how much data (e.g., historical scenarios) are needed to ensure optimality with a high level of confidence. Inference techniques used in SP are thus similar in spirit to those used in parameter estimation, in which one seeks to determine if existing experimental data is sufficient to achieve a desired degree of confidence in the parameters.

To explain how inference analysis techniques work, we first define the exact cost function $q(\pi) := \mathbb{E}_\Xi[\varphi(\pi, \Xi)]$ and use the convention that $q(\pi)=M$ if $\pi \notin \Pi(\xi)$ for at least one ξ and where $M \in \mathbb{R}_+$ is a large number. In other words, if the controller does not satisfy the constraints for a given realization ξ, we assign a large penalty. Since $q(\pi)$ can never be evaluated (because of the expectation operator), we will use Monte Carlo sampling to create statistical estimators of the form $\hat{q}_N(\pi) := N^{-1}\sum_{i=1}^{N} \varphi(\pi, \xi_i)$. Here, the realizations ξ_i, $i=1, \ldots,$

N are assumed to be i.i.d. (independent and identically distributed). In other words, the realizations are generated at random.

We now define candidate solution π, which can be obtained by solving problem (0.13) using SAA with a finite number of scenarios generated from Monte Carlo sampling or from historical data. To evaluate the optimality of this candidate solution, consider now $j=1,\ldots,T$ data batches each with $i=1,\ldots,\bar{N}$ i.i.d. data realizations and denote the realizations $\xi_{i,j}$. We use these batches to obtain the mean estimator $\bar{q}_T(\pi)=T^{-1}\sum_{j=1}^{T}\hat{q}_{\bar{N}}^{j}(\pi)$ with $\hat{q}_{\bar{N}}^{j}=\bar{N}^{-1}\sum_{i=1}^{\bar{N}}\varphi(\pi,\xi_{i,j})$ for all $j=1,\ldots,T$.

From a controller tuning stand-point, we interpret the evaluation of $\hat{q}_{\bar{N}}^{j}(\pi)$ as *testing* the controller settings under different operational scenarios selected at random. Since $\xi_{i,j}$ are i.i.d, we have that the estimator $\hat{q}_{\bar{N}}^{j}(\pi)$ is unbiased and thus $\mathbb{E}[\hat{q}_{\bar{N}}(\pi)]$. Furthermore, from the central limit theorem we have that the estimator is also normally distributed with mean $\lim_{T\to\infty}\bar{q}_T(\pi)=\mathbb{E}[\hat{q}_{\bar{N}}^{j}(\pi)]=q(\pi)$.

Because the candidate solution π is at best optimal, we have that $q(\pi)\geq q(\pi^*)$ for all π. Consequently, $\hat{q}_T(\pi)$ is the mean of the estimator of the upper bound $\hat{q}(\hat{\pi})$. Moreover, because the estimator is normally distributed, we can construct a confidence interval for $q(\pi)$ of the form:

$$\mathbb{P}\left(\bar{q}_T(\pi)-z_{\alpha/2}\frac{\bar{s}_T(\pi)}{\sqrt{T}}\leq q(\pi)\leq\bar{q}_T(\pi)+z_{\alpha/2}\frac{\bar{s}_T(\pi)}{\sqrt{T}}\right)=1-\alpha, \tag{0.14}$$

where $\bar{s}_T(\pi)=\sqrt{\frac{1}{T-1}\sum_{j=1}^{T}(\hat{q}_{\bar{N}}^{j}(\pi)-\bar{q}_T(\pi))^2}$ is the estimator variance and $z_{\alpha/2}$ is the $1-\alpha/2$ quantile of the standard normal distribution. For $\alpha=0.05$ (95% confidence interval) we have $z_{\alpha/2}=1.96$.

To estimate a lower bound for the candidate solution we create an estimator of the form $\underline{q}_T=T^{-1}\sum_{i=1}^{T}\min_{\pi}\hat{q}_{\bar{N}}^{i}(\pi)$. Here, the estimator is evaluated using $i=1,\ldots,T$ i.i.d data batches that are different from those used to estimate the statistics of the upper bound. From the central limit theorem we have that $\lim_{T\to\infty}\underline{q}_T(\pi)=\mathbb{E}[\hat{q}_{\bar{N}}^{j}(\pi)]=q(\pi)$ for any π. Moreover, we have that $q(\pi)=\mathbb{E}[\hat{q}_{\bar{N}}^{j}(\pi)]\geq\mathbb{E}[\min_{\pi}\hat{q}_{\bar{N}}(\pi)]$ for any π. This implies that $q(\pi^*)\geq\mathbb{E}[\hat{q}_{\bar{N}}^{j}(\pi)]$. Consequently, we can construct a confidence interval for the lower bound of the form:

$$\mathbb{P}\left(\underline{q}_T-z_{\alpha/2}\frac{\underline{s}_T}{\sqrt{T}}\leq\mathbb{E}[\min_{\pi}\hat{q}_{\bar{N}}(\pi)]\leq\underline{q}_T+z_{\alpha/2}\frac{\underline{s}_T}{\sqrt{T}}\right)=1-\alpha, \tag{0.15}$$

where $\underline{s}_T=\sqrt{\frac{1}{T-1}\sum_{j=1}^{T}(\min_{\pi}\hat{q}_{\bar{N}}^{j}(\pi)-\underline{q}_T)^2}$. From a controller tuning stand-point, we interpret the evaluation of the lower bound as *finding different optimal controller settings for different sets of data batches*. Clearly, however, we ultimately only want *a single set of controller settings* that work for all scenarios. Consequently, the evaluation of the lower bound is only performed to quantify *ideal* performance.

When the candidate solution π is computed from (0.13) using a sufficiently large number of scenarios, we expect that the variance of the lower bound is small (i.e., changing the data batch will not affect the tuning specifications and cost). We also expect that the upper bound variance will shrink when a large number of scenarios are used and that the lower and upper bounds will get close to each other. In other words, if the controller tuning settings are obtained with a sufficiently large number of scenarios, we have a *guarantee that the likelihood of finding a better set of specifications will be small* (the confidence interval of the difference of the upper and lower bound will be small). In a data-based setting, inference analysis capabilities provide a mechanism to determine when adding more scenario information is no longer helping improve the robustness and performance of the controller settings.

Sparse grids

Sparse grids are computationally tractable techniques for approximating multi-dimensional integrals and have been used to approximate expectation operators in SP.[27–29] Sparse grids are constructed by extending one-dimensional quadrature rules to higher dimensions via a truncated tensor product. This truncation helps overcome the curse of dimensionality. With discretization level ℓ and mesh size $h_\ell:=2^{-\ell}$, sparse grid approaches require $\mathcal{O}(h_\ell^{-1}\cdot\log(h_\ell^{-1})^{d-1})$ nodes (i.e., grid points) to approximate a smooth function over a d-dimensional domain with accuracy $\mathcal{O}(h_\ell^{2}\cdot\log(h_\ell^{-1})^{d-1})$. In contrast, conventional full grid methods require $\mathcal{O}(h_\ell^{-d})$ nodes for accuracy $\mathcal{O}(h_\ell^{2})$.[30]

A central assumption used in sparse grids is that the weight function of the quadrature can be decomposed into independent weight functions for each dimension. For the controller tuning problem, this would imply that the random variables are independent. If there exist correlations in Gaussian variables, one can still apply the technique by transforming the space using an eigenvalue decomposition. Decomposition for other types of distributions, however, is not trivial and one often must assume independence. This is not a strong assumption if no data exists to begin with and thus one needs to design controllers over simple domains (e.g., by assuming uniform distributions).

A limitation of using sparse grids in an SP context is that the quadrature weights might be negative and this might lead to performance issues when approximating expectation operators (e.g., unbounded costs).[27] However, sparse grids are still a highly useful scenario generation technique because the grid (quadrature) points are selected in a way that they cover the integration domain with few points (hence the name sparse grid). Such *domain covering* property is particularly attractive in a controller design context when one is interested in exploring *worst-case (robust) formulations* that only seek to explore the boundaries of performance of the controller (which often lie on the boundaries of the domain of the data). We recall that worst-case SP formulations do not involve expectation operators (see (0.10)) and thus the presence of negative weights is not relevant. We also highlight that Monte Carlo might require a very large number of samples to cover the domain (see Figure 1). Monte Carlo, however, is more flexible than sparse grids in that it can capture different types of distributions.

In this work, we use sparse grids to efficiently cover the domain and we use Clenshaw curtis exponential growth (CCEG) sparse grids.[31] We have found that these grids work better than grids derived from Gaussian quadrature rules. This is because CCEG grids place more points along the domain boundary and thus better capture extreme events. The proposed sparse grid formulations are motivated by interesting results from traditional control theory. For instance, Bode's integral theorem states that it is always possible to find a disturbance that destabilizes a PID controller.[32] The sparse grids

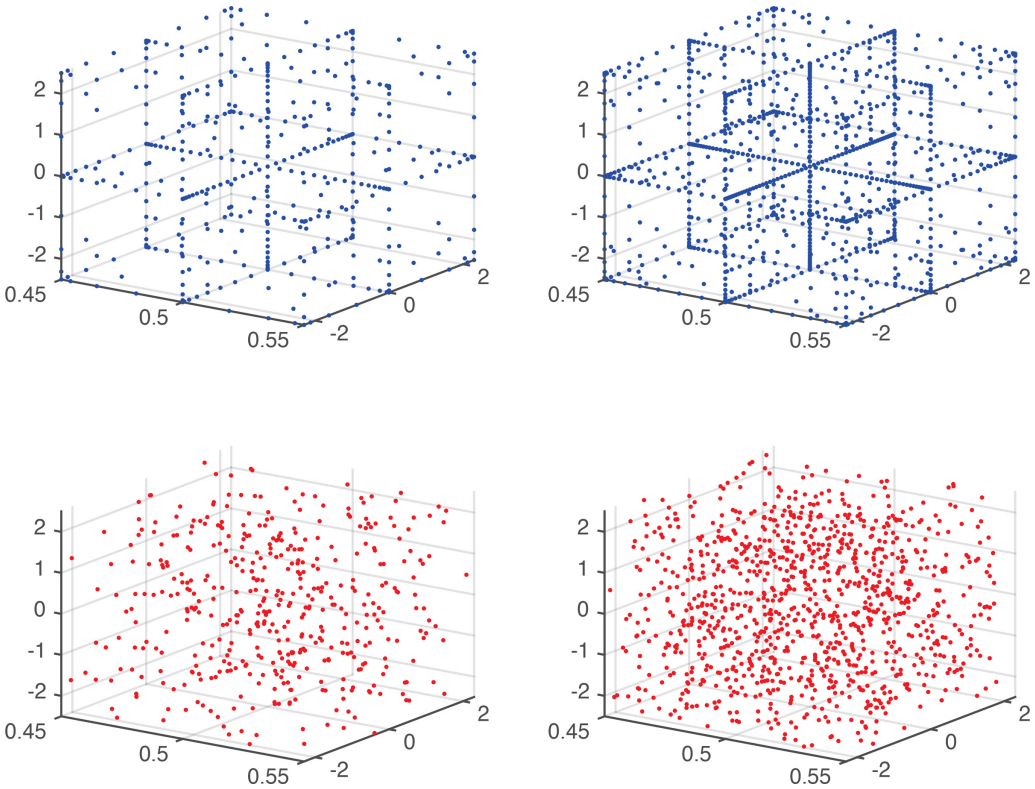

Figure 1. Sparse grid (top) and Monte Carlo samples (bottom).

Left graphs contain 441 scenarios (sparse grid of level ℓ=5 and the right graphs contain 1073 scenarios (sparse grid of level ℓ=6). [Color figure can be viewed at wileyonlinelibrary.com]

approach thus provides a systematic mechanism to find such a disturbance.

Computational Tools

In this section we present tools that enable the implementation and solution of large-scale controller tuning problems. We begin by noticing that the SP problem (0.13) can be expressed as:

$$\min_{\pi\in\Pi, y_\xi\in Y_\xi} \sum_{\xi\in\Omega} f_\xi(\pi, y_\xi) \tag{0.16a}$$

$$\text{s.t. } c_\xi(\pi, y_\xi)=0,\ \xi\in\Omega. \tag{0.16b}$$

Here, $y_\xi \in Y_\xi := \{y_\xi | \underline{y}_\xi \le y_\xi \le \bar{y}_\xi\}$ are the variables in scenario $\xi \in \Omega$ (associated to the dynamic model and operational constraints) and $\pi \in \Pi := \{\pi | \underline{\pi} \le \pi \le \bar{\pi}\}$ are the controller settings. The above problem can also be expressed in the equivalent *lifted form*:

$$\min_{\{\pi_\xi\}\in\Pi, y_\xi\in Y_\xi} \sum_{\xi\in\Omega} f_\xi(\pi_\xi, y_\xi) \tag{0.17a}$$

$$\text{s.t. } c_\xi(\pi_\xi, y_\xi)=0,\ \xi\in\Omega \tag{0.17b}$$

$$\pi_{\xi+1}=\pi_\xi,\ \xi\in\Omega\backslash\{|\Omega|\}. \tag{0.17c}$$

Here, the coupling variables π have been replicated into $|\Omega|$ copies π_ξ. The linking constraints (0.17c) are also known as the *non-anticipativity constraints*. This partially-separable structure highlights that the SP becomes fully decoupled when the linking constraints are removed. The fully decoupled structure can be used when evaluating lower bounds for inference analysis and captures the idealized situation in which different controller tuning settings π_ξ can be chosen in each scenario $\xi \in \Omega$.

With some abuse of notation, (0.16) can also be written in the following recourse form:

$$z(\Pi)= \min_{\{\pi_\xi\}\in\Pi} \sum_{\xi\in\Omega} Q_\xi(\pi_\xi), \quad \text{s.t.} \quad \pi_{\xi+1}=\pi_\xi,\ \xi\in\Omega\backslash\{|\Omega|\}, \tag{0.18}$$

where the recourse function for each scenario $\xi \in \Omega$ is:

$$Q_\xi(\pi) := \min_{y_\xi\in Y_\xi} f_\xi(\pi, y_\xi) \quad \text{s.t. } c_\xi(\pi, y_\xi)=0. \tag{0.19}$$

The partially-separable structure of the controller tuning problem can be exploited to enable highly efficient solution computations. The tuning problem, however, is nonconvex (even when the dynamic model is linear) and we can thus search for local solutions in complex models using gradient-based algorithms (such as interior point solvers) or for global solutions in simpler models (using specialized branch and bound techniques). The approach followed to exploit structure varies significantly in local and global algorithms, as we explain next.

Local search solvers

When we are interested in finding *local solutions*, one can use an interior point solver and exploit the underlying linear algebra when computing Newton (search) steps.[33–35] It is well-known that the linear algebra system of problem (0.16) (a similar approach can be applied to (0.17)) can be permuted to the following block-bordered-diagonal (BBD) form:

$$\begin{bmatrix} K_\pi & B_1^T & B_2^T & \dots & B_{|\Omega|}^T \\ B_1 & K_1 & & & \\ B_2 & & K_2 & & \\ \vdots & & & \ddots & \\ B_{|\Omega|} & & & & K_{|\Omega|} \end{bmatrix} \begin{bmatrix} \Delta w_\pi \\ \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_{|\Omega|} \end{bmatrix} = - \begin{bmatrix} r_\pi \\ r_1 \\ r_2 \\ \vdots \\ r_{|\Omega|} \end{bmatrix}, \quad (0.20)$$

where $\Delta w_\pi = (\Delta\pi, \Delta\lambda_\pi)$, $\Delta w_\xi = (\Delta y_\xi, \Delta\lambda_\xi)$ are the Newton steps,

$$K_\pi = W_\pi \quad (0.21a)$$

$$K_\xi = \begin{bmatrix} W_\xi & J_\xi^T \\ J_\xi & \end{bmatrix}, \quad (0.21b)$$

$$B_\xi = \begin{bmatrix} R_\xi \\ T_\xi \end{bmatrix}, \quad (0.21c)$$

$J_\xi = \nabla_{y_\xi} c_\xi(\pi, y_\xi)$, $T_\xi = \nabla_\pi c_\xi(\pi, y_\xi)$, $W_\pi = \mathrm{diag}(\pi)^{-1}\mathrm{diag}(\lambda_\pi)$, $W_\xi = \nabla_{y_\xi, y_\xi}\mathcal{L} + \mathrm{diag}(y_\xi)^{-1}\mathrm{diag}(\lambda_\xi)$, $R_\xi = \nabla_{\pi, y_\xi}\mathcal{L}$, $\mathcal{L}(\cdot)$ is the Lagrange function of (0.16), λ_π are the multipliers of $\pi \in \Pi$ and λ_ξ are the multipliers of $y_\xi \in Y_\xi$.

The BBD system (0.20) can be solved in parallel by using a Schur complement decomposition approach. This requires the solution of the following systems:

$$\left(K_\pi - \sum_{\xi\in\Omega} B_\xi^T K_\xi^{-1} B_\xi\right) \Delta w_\pi = -r_\pi + \sum_{\xi\in\Omega} B_\xi^T K_\xi^{-1} r_\xi \quad (0.22a)$$

$$K_\xi \Delta w_\xi = -r_\xi - B_\xi \Delta w_\pi, \ \xi \in \Omega. \quad (0.22b)$$

Here, $S := K_\pi - \sum_{\xi\in\Omega} B_\xi^T K_\xi^{-1} B_\xi$ is the *Schur complement matrix*.

Global search solvers

When we are interested in finding *global solutions*, one needs to resort to techniques that exploit structure at the problem formulation level. In the context of stochastic programs, one can construct spatial branch and bound (B&B) algorithms that progressively explore the feasible set π. At each node in the spatial B&B scheme we use a restriction of the set Π and seek to find upper and lower bounds for problem (0.18).

If the non-anticipativity constraints of (0.18) are removed, we obtain the lower bounding problem of the form:

$$\beta(\Pi) := \min_{\{\pi_\xi\}\in\Pi} \sum_{\xi\in\Omega} Q_\xi(\pi_\xi). \quad (0.23)$$

Clearly, this lower bounding problem can be decomposed into $|\Omega|$ subproblems of the form:

$$\beta_\xi(\Pi) := \min_{\pi_\xi\in\Pi} Q_\xi(\pi_\xi), \quad (0.24)$$

It is also obvious that $\beta(\Pi) \le z(\Pi)$ because the feasible region of (0.18) is a subset of the feasible region of (0.23). Moreover, we have that $\beta(\Pi_1) \ge \beta(\Pi_2)$ for $\Pi_1 \subset \Pi_2$. The lower bounding problem is also called the wait and see problem, and the gap between the primal problem (0.18) and the lower bounding problem (0.23) is called the expected value of perfect information (EVPI). We highlight that $\beta_\xi(\Pi)$ is obtained by solving the scenario subproblems to *global optimality*.

An upper bound for the stochastic program (0.18) can be obtained by fixing the first stage variable at a candidate solution $\hat{\pi} \in \Pi$. The upper bound is denoted as $\gamma(\Pi) := \sum_{\xi\in\Omega} Q_\xi(\hat{\pi})$ and we note that the upper bound can be computed by solving $|\Omega|$ subproblems. These subproblems are also solved to *global optimality*. It is easy to see that $z(\Pi) \le \gamma(\Pi)$ holds for any $\hat{\pi} \in \Pi$. When the *absolute gap* $\beta(\Pi) - \gamma(\Pi)$ is zero (or sufficiently small), we have found a global solution for problem (0.18). A B&B scheme thus proceeds to sequentially shrink the set Π until a global solution is found. One also often monitors the relative gap $(\beta(\Pi) - \gamma(\Pi))/\gamma(\Pi)$.

Global solutions of the controller tuning problem can be found efficiently by exploiting the partially-separable structure of the problem (compared to using off-the-shelf global solvers). This is because the dimension of the coupling variables π is small and because the solution of the subproblems can proceed in parallel. Such an approach is implemented in the solver SNGO.[36] We also have that, if the dynamic model is linear, the upper bounding subproblems are convex problems and thus can be solved efficiently. The lower bounding scenario subproblems, however, remain nonconvex because we need to simultaneously handle the controller settings and the dynamic model response. This observation highlights that the decomposition scheme is limited by the scope of off-the-shelf global optimization solvers (such as SCIP[37] and BARON[38]) (which are needed to solve the scenario subproblems). We also highlight that a global optimization framework for SPs can also exploit more advanced techniques that handle differential equations in a simulation-based setting.[39] We do not explore such techniques here but highlight that these provide an interesting future research direction.

Modeling languages

Modeling languages such as Plasmo (https://github.com/jalving/Plasmo.jl.git) can be used to easily express the SPs under study. Plasmo is a Julia-based algebraic modeling framework that facilitates the construction and analysis of large-scale structured optimization models. To do this, it leverages a hierarchical graph abstraction wherein nodes and edges can be associated with individual JuMP[40] scenario problems that are linked together (e.g., by using non-anticipativity constraints).[41,42] Given a graph structure with models and connections, Plasmo can produce either a pure (flattened) optimization model to be solved using off-the-shelf optimization solvers such as IPOPT[43] or it can communicate graph structures to structure-exploiting solvers such as PIPS-NLP[33] and SNGO.[36] Modeling and solution capabilities for stochastic programs are also provided by the popular Python-based package PySP.[44]

The code snippet shown in Figure 2 illustrates how to implement the PID tuning problem in Plasmo while in Figure 3 we illustrate how to generate an individual JuMP scenario subproblem. As can be seen, the individual scenario models are created and appended to the parent node on-the-fly to create a two-level graph structure and the variables are linked using non-anticipativity (linking) constraints. From the snippet we can also see how Plasmo can create a flattened NLP to be solved by off-the-shelf solvers like IPOPT or can call PIPS-NLP (or SNGO) to solve the problem in parallel. This allows the user to compare computational performance.

```
using IPOPT, Plasmo, JuMP, Distributions

MPI.Init() # Initialize MPI

NS =10 # Number of scenarios
N=100 # Number of timesteps
T=collect(1:N) # Set of times
S=collect(1:NS) # Set of scenarios

# get scenarios for model gain and set-points
dis = Normal(3,1)
  K = rand(dis,NS)
dis = Uniform(-2.3,2.3)
xsp = rand(dis,NS)

include("get_scenario_model.jl") #scenario model building function

# create two-stage graph moddel
PID=GraphModel()
master = Model()
master_node = add_node(PID,master)

# add variables to parent node
@variable(master, KP)
@variable(master, KI)
@variable(master, KD)

# create array of children models
PIDch=Array{NodeOrEdge}(NS)
for s in S
        # get JuMP scenario model
        bl = get_scenario_model(s)
        child = add_node(PID,bl)
        # add children to parent node
        PIDch[s] = child
        # link children to parent variables
        @linkconstraint(PID, bl[:KP]==KP)
        @linkconstraint(PID, bl[:KI]==KI)
        @linkconstraint(PID, bl[:KD]==KD)
end

# solve with PipsNlp
pipsnlp_solve(PID,master_node,PIDch)

# solve with IPOPT
IPOPT_solve(PID)

MPI.Finalize() # Finalize MPI
```

Figure 2. Snippet of a PID Tuning implementation in Plasmo.

[Color figure can be viewed at wileyonlinelibrary.com]

Numerical Case Studies

In this section we demonstrate the concepts discussed using a couple of numerical examples. We first use a single-loop PID tuning example to illustrate concepts on inference analysis, risk measures and dominance, and local/global optimization. We then use a multi-loop PID tuning example for a distillation column to demonstrate scalability of modern tools.

Expected value and robust settings

We consider an optimal tuning formulation for a first-order linear system:

$$\min_{K_P,K_I,K_D,\nu} \mathbb{E}_{\Xi}\left[\nu+\alpha^{-1}[\varphi(\epsilon_{\mathcal{T}},u_{\mathcal{T}},\Xi)-\nu]_+\right] \tag{0.25a}$$

$$\tau(\xi)\dot{x}(t,\xi)+x(t,\xi)=K_u(\xi)u(t)+K_d(\xi)d(t,\xi), \quad t\in\mathcal{T},\xi\in\Omega \tag{0.25b}$$

$$\epsilon(t)=\bar{x}(t,\xi)-x(t,\xi), \quad t\in\mathcal{T},\xi\in\Omega \tag{0.25c}$$

$$u(t)=K_P\epsilon(t,\xi)+K_I\dot{I}(t,\xi)+K_D\dot{\epsilon}(t,\xi), \quad t\in\mathcal{T},\xi\in\Omega \tag{0.25d}$$

$$\dot{I}(t,\xi)=\epsilon(t), \quad t\in\mathcal{T},\xi\in\Omega \tag{0.25e}$$

$$|x(t,\xi)|\le\bar{x}, \quad t\in\mathcal{T}\ \xi\in\Omega \tag{0.25f}$$

$$|u(t,\xi)|\le\bar{u}, \quad t\in\mathcal{T}\ \xi\in\Omega \tag{0.25g}$$

$$|\epsilon(T,\xi)|\le\bar{\epsilon}, \quad \xi\in\Omega \tag{0.25h}$$

$$0\le K_P\le\bar{K}_P \tag{0.25i}$$

$$0\le K_I\le\bar{K}_I \tag{0.25j}$$

$$0\le K_D\le\bar{K}_D \tag{0.25k}$$

$$x(0,\xi)=x_0, \quad \xi\in\Omega, \tag{0.25l}$$

where,

```
function get_scenario_model(s)

m=Model()
# variables (states and inputs)
@variable(m,-2.5<=x[T]<=2.5)
@variable(m,-2.0<=u[T]<=2.0)
@variable(m, inte[T])
@variable(m, de[T])
# variables (controller design)
@variable(m, 0<=KP<=10)
@variable(m, 0<=KI<=100)
@variable(m, 0<=KD<=1000)
# dynamic constraints
@constraint(m, [t in Tm],tau*(x[t+1]-x[t])/h + x[t+1]== K[s]*u[t+1]+Kd*d)
@constraint(m, [t in Tm], u[t+1] == KP*e[t+1]+ KI*inte[t+1] + KD*de[t+1])
@constraint(m, [t in Tm], (inte[t+1]-inte[t])/h == e[t+1])
@constraint(m, [t in T], e[t] == xsp[s]-x[t])
@constraint(m, [t in T], de[t] == (e[t+1]-e[t])/h)
@constraint(m, x[1] == x0[s])
@constraint(m, inte[1] == 0)
# objective function
@objective(m, Min, (1/(N*NS))*sum(e[t]^2 + u[t]^2 for t in T))

return m
end
```

Figure 3. Snippet of JuMP scenario model.

[Color figure can be viewed at wileyonlinelibrary.com]

$$\varphi(\epsilon_{\mathcal{T}}, u_{\mathcal{T}}, \xi) = \int_{\mathcal{T}} (\gamma_\epsilon \epsilon(t,\xi)^2 + \gamma_u u(t,\xi)^2) dt. \tag{0.26}$$

We set the random data to $K_u(\xi) \sim \mathcal{N}(3, 0.3)$, $1/\tau(\xi) \sim \mathcal{N}(0.5, 0.05)$, $\bar{x}(t,\xi) \sim \mathcal{U}(-2.3, 2.3)$ for all $t \in \mathcal{T}$, $d(t,\xi) \sim \mathcal{U}(-2.5, 2.5)$ for all $t \in \mathcal{T}$, $K_d(\Xi) = 0.5$, and $x_0 = 0$. We also use the bounds $\bar{K}_P = 100$, $\bar{K}_I = 1000$, $\bar{K}_D = 1000$, $\bar{\epsilon} = 0.1$, $\bar{x} = 20$, and $\bar{u} = 20$. The dynamical model was discretized using an implicit Euler discretization scheme with $N = 100$ time steps and we use $|\Omega| = 1000$ scenarios. This formulation has 401,004 variables and four degrees of freedom and can be solved with IPOPT in 124 s (around 2 min).

We begin our discussion by comparing the performance of expected value and robust (worst-case) variants of formulation (0.25). We recall that the expected value formulation is obtained by setting $\alpha = 1$ while the robust formulation is obtained by setting $\alpha \to 0$. The optimal control settings obtained with the expected value formulation are $\pi^* = (10, 53.88, 0)$ while the robust formulation gives $\pi^* = (10, 31.20, 0)$. As can be seen, the proportional gain K_P hits the upper bound in both cases and the derivative gain K_d is zero. A significant difference is obtained in the integral gain K_I. We have also found that the worst-case cost of the expected value formulation is 100.37 while that achieved with the robust formulation is 100.27. Clearly, the impact of the robust formulation is not that marked from this stand-point. From an actual performance stand-point, however, we found that the closed-loop response of the controller is significantly different under both formulations. In Figure 4 we present the closed-loop response for the tracking error $\epsilon_{\mathcal{T}}(\xi)$ and we see that the robust controller gives a smoother transition. It is also remarkable that both optimal controller settings are able to bring the closed-loop error to the origin in all the 1000 scenarios considered (which cover a wide range of simultaneous uncertainties). This illustrates that highly reliable PID controllers can be designed by using an SP setting.

Inference analysis

We now analyze the performance of the expected value controller using inference analysis techniques as we increase the number of scenarios. The results are presented in Figure 5. Here, we can see that significant variance is obtained in the upper and lower bounds of the cost function and in the integral gain K_I when the number of scenarios are less than 100 (the proportional and derivative gain where 10 and 0, respectively,

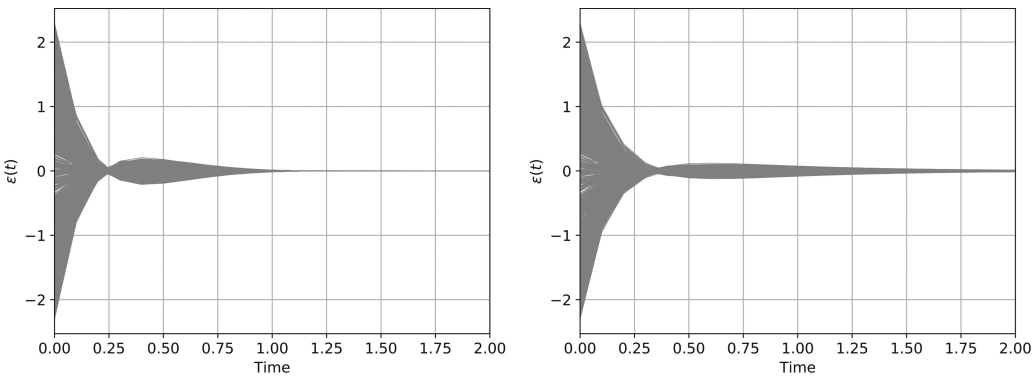

Figure 4. Closed-loop response for state with expected value (left) and robust (right) for PID settings (1000 scenarios).

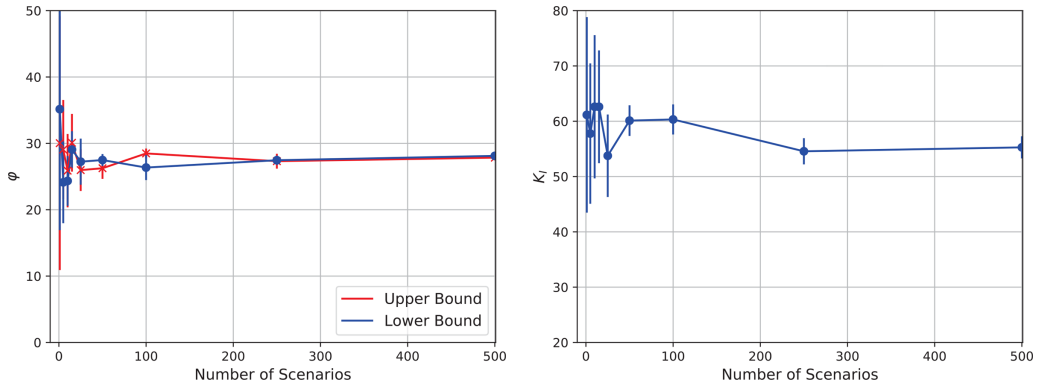

Figure 5. Upper and lower bounds (left) and variance of K_I (right) with increasing number of scenarios.

[Color figure can be viewed at wileyonlinelibrary.com]

in all cases). We can also observe that the integral gain settles after around 250 scenarios and that the upper and lower bounds converge. This indicates that no further improvement can be obtained by adding more scenarios. From a practical stand-point this would indicate that there is a *statistical guarantee* that, no matter how many more operational scenarios are considered, there is high confidence that the controller settings are already optimal (in terms of the expected cost). The results also highlight the danger of tuning controllers using just a few operational scenarios, because these will likely lead to poor performance when confronted with operational scenarios not considered in the tuning.

Stochastic dominance

We now illustrate the performance of the controller from a stochastic dominance stand-point. In particular, we aim to illustrate that, by minimizing CVaR, there is a guarantee that no other controller settings can be found that have a better cost CDF. In Figure 6 we compare the probability distributions for the cost when computing the controller settings using CVaR with $\alpha=0.01$ (we minimize the average over the 1% tail) and when the optimal controller settings are perturbed 50% (to give an alternative and suboptimal controller). We can see the distribution obtained by minimizing CVaR has a thinner tail, which indicates that it will perform better in extreme scenarios than the suboptimal controller. However, the shape of the distributions is not remarkably different and thus this is hard to appreciate by pure visual inspection. The difference becomes clearer when one compares the CDF of the costs, which are shown in Figure 7. We can see that the CDF of the optimal controller settings dominates the one of the suboptimal alternative to first order (for every v). We recall, however, that first-order dominance is not guaranteed to occur in general (only second order dominance is guaranteed). Nevertheless, by comparing the CDFs, it becomes clear that the optimal settings give much better performance and that the highest difference is obtained in a cost range of 60–80 (such difference can be verified by comparing the probability densities in this cost range in Figure 6). Similarly, from the CDFs it becomes clear that there is not much difference in the distributions in the cost range of 0–20.

Local vs. global solutions

We now proceed to certify that the controller settings obtained with IPOPT are globally optimal (we recall that this is a local solver so there is no guarantee that the solution is globally optimal). To do this, we solve expected value problems with increasing number of scenarios with SNGO (which exploits the nearly separable structure of the problem) and with the off-the-shelf global solver SCIP.[37] The results are presented in Table 1. The results confirm that the controller settings obtained with IPOPT are globally optimal in all cases (within an absolute gap tolerance of 0.01). We also see that SNGO can prove global optimality significantly faster compared to SCIP (in some instances SCIP cannot provide an upper bound within 12 h). This is because SNGO decomposes the problem into scenario subproblems. In this particular instance, global optimality is detected by SNGO at the first branch and bound (B&B) node while SCIP needs to explore thousands of nodes. We also highlight that the solution time of

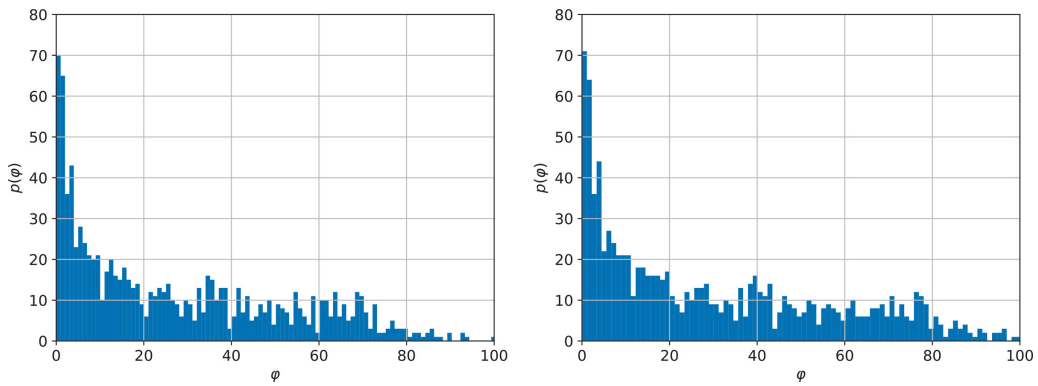

Figure 6. Cost histogram for CVaR solution with $\alpha=0.01$ (left) and for suboptimal alternative (1000 scenarios).

[Color figure can be viewed at wileyonlinelibrary.com]

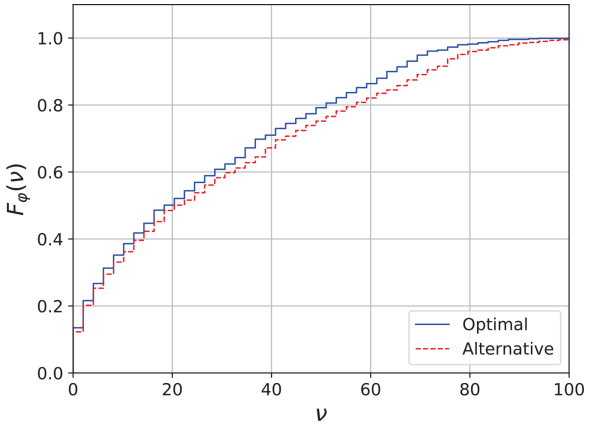

Figure 7. Stochastic dominance of optimal CVaR solution with α=0.01 over suboptimal alternative (1000 scenarios).

[Color figure can be viewed at wileyonlinelibrary.com]

SNGO grows linearly with the number of scenarios (because the solver runs in serial). Parallelization can potentially bring down the solution time to around 15 s.

We designed a more difficult tuning experiment with a nonlinear dynamical model of the form:

$$\tau(\xi)\dot{x}(t,\xi)+x(t,\xi)^2=K_u(\xi)u(t)+K_d(\xi)d(t,\xi), \quad t\in\mathcal{T},\xi\in\Omega. \tag{0.27}$$

The results for the SP tuning formulation using the nonlinear dynamic model are summarized in Table 2. We can see that this problem can be solved within a relative gap of 1% with SNGO significantly faster than with SCIP. It is clear, however, that this problem is more difficult to solve. The solution time of SNGO again scales linearly but parallelization can be bring the time down to around 90 seconds. In this case, we found a significant improvement over IPOPT of nearly 17% for a tuning problem with 10 scenarios. As we increase the number of scenarios, however, no improvement can be achieved. This indicates that IPOPT identifies global solutions as we increase the number of scenarios. This is an interesting phenomenon that we are currently investigating. We hypothesize that, by adding more scenarios, the tuning problem becomes better defined and with this local optima disappear. This is because, in the presence of more scenarios, there are less combinations of controller settings that can satisfy the constraints over all scenarios (while when few scenarios are used many more combinations can satisfy such constraints).

Sparse grids and Monte Carlo

We now compare the efficiency of sparse grids and Monte Carlo sampling at identifying worst-case cost solutions for the controller. To do so, we solved a robust version of the tuning problem with sparse grids and Monte Carlo for increasing number of scenarios. In this study, we relaxed the upper bound of the proportional gain to $\bar{K}_P$=100 and we use the scenarios shown in Figure 1 (for $|\Omega|$=441 and $|\Omega|$=1073). The results are summarized in Table 3. As can be seen, sparse grids can identify the worst-case cost with 69 scenarios (if the number of scenarios is increased the tuning parameters and the worst-case cost do not change). Monte Carlo sampling finds increasingly worse solutions as we increase number of scenarios as well but cannot identify the worst-case solution even with 1073 scenarios. In addition, the tuning parameters do not settle. This illustrates that Monte Carlo is not particularly effective at placing scenarios in the boundary of the uncertainty

Table 1. Computational Performance of SNGO and SCIP on Tuning Problem with Linear Model

Problem		SNGO					SCIP					IPOPT	
Ω	Var	Time	rGap (%)	LB	UB	B&B	Time	rGap (%)	LB	UB	B&B	Time [s]	Obj
1	404	15.64	2.67	0.31	0.31	1	1.19	1.68	0.31	0.31	1	0.24	0.31
5	2008	19.83	2.86	0.22	0.22	1	22.51	2	0.22	0.22	1	0.34	0.22
10	4013	29.37	2.84	0.35	0.36	1	334.7	2	0.35	0.36	2000	0.54	0.36
50	20053	111.1	2.52	0.31	0.31	1	43200		0.27	–	3300	1.69	0.31
100	40103	192.81	3.63	0.26	0.27	1	43200		0.26	–	–	3.2	0.27
1000	401003	2789	2.67	0.27	0.27	1	43200		0.23	–	–	49	0.27

Table 2. Computational Performance of SNGO and SCIP on Tuning Problem with Nonlinear Model

	SNGO			SCIP			IPOPT
Ω	Time	rGap (%)	B&B	Time	rGap (%)	B&B	% Improv.
10	942	1	58	43200	1.29	2248314	17.1
20	1330	1	37	27045	1.00	829181	0.0
40	3862	1	39	43200	2.16	595294	0.0

Table 3. Worst-Case Solutions Obtained with Sparse Grids and Monte Carlo

	Sparse grids				Monte Carlo			
Ω	$\max_{\xi\in\Omega}\varphi(\xi)$	K_P	K_I	K_D	$\max_{\xi\in\Omega}\varphi(\xi)$	K_P	K_I	K_D
25	87.46	199.97	1.90	6.24	71.34	199.94	1.45	6.76
69	90.94	199.97	1.51	7.39	72.73	199.58	0.77	7.10
177	90.94	199.97	1.51	7.39	77.84	199.96	1.77	6.16
441	90.94	199.97	1.51	7.39	84.83	199.96	1.37	7.37
1073	90.94	199.97	1.51	7.39	86.31	199.97	1.66	6.80

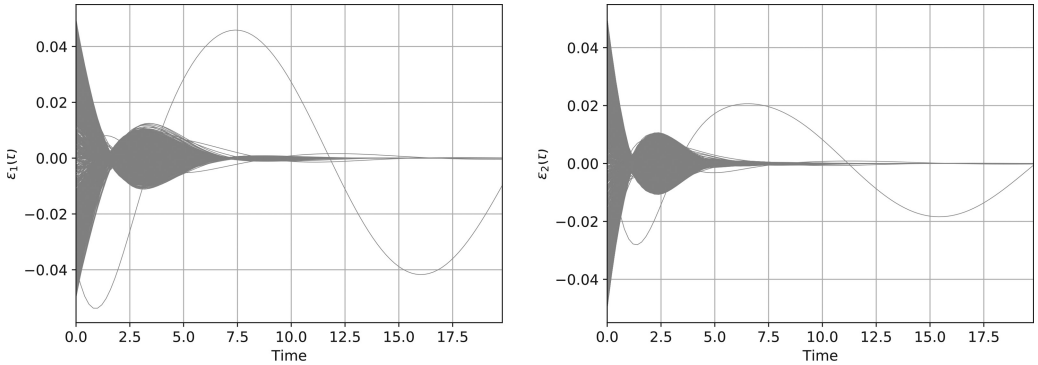

Figure 8. Closed-loop response of distillate (left) and bottoms (right) methanol composition under $|\Omega|=1000$ scenarios.

domain. We highlight, however, that Monte Carlo sampling is a more flexible approach to handle complex distributions.

PID controller tuning as constrained MPC

The proposed optimal PID controller tuning formulation is a restricted form of MPC. This is because the PID controller restricts the control actions to follow a certain structural form. Consequently, one would expect that even an optimally tuned PID controller will not be capable of achieving the flexibility and performance of an MPC controller. The proposed controller tuning formulation can easily be modified to assess the hypothetical ideal performance of MPC and with this evaluate how far is the performance of optimally tuned PID controllers. To illustrate this, we run an optimal PID controller tuning problem with 1000 Monte Carlo scenarios to minimize the expected cost and found that the optimal cost is $\varphi=68.36$. The performance of the hypothetical MPC controller is $\varphi=26.29$ (an improvement of 61%). Clearly, the MPC controller performance is drastically superior. We emphasize, however, than in certain applications it is difficult to deploy MPC controllers (due to computing infrastructure or cost constraints). Nevertheless, we believe that it should always be of practical interest to evaluate how far the performance of an *optimally* tuned PID controller is to that of MPC because such a gap is not always obvious. In recent work for controller tuning for wind turbines, for instance, we found that optimally-tuned settings improve wind power by 17% (compared to nominal settings). MPC improves performance of the optimally tuned controllers by 20%.[3] Unfortunately, deploying MPC in wind turbines is technically challenging but this is a topic that industry is investigating. The proposed optimal controller tuning framework thus provides an *intermediary step* toward realizing economic benefits (while MPC deployments are investigated).

Scalability

In our last set of numerical experiments we aim to demonstrate that state-of-the-art solvers can handle large-scale tuning problems. We use a dynamic model for the experimental binary water-methanol distillation from Wood and Berry.[45] In the frequency domain, the input-output transfer function system is:

$$\begin{bmatrix} y_1(j\omega) \\ y_2(j\omega) \end{bmatrix} = \begin{bmatrix} \dfrac{12.8e^{-j\omega}}{16.7(j\omega)+1} & \dfrac{-18.9e^{-3j\omega}}{21.0(j\omega)+1} \\ \dfrac{-6.6e^{-7j\omega}}{10.9(j\omega)+1} & \dfrac{-19.4e^{-3j\omega}}{14.4(j\omega)+1} \end{bmatrix} \begin{bmatrix} u_1(j\omega) \\ u_2(j\omega) \end{bmatrix} \tag{0.28}$$

Here, $y_1(\cdot)$ is the methanol concentration in the top of the column, $y_2(\cdot)$ is the methanol concentration in the bottom of the column, $u_1(\cdot)$ is the recycle reflux flow rate and $u_2(\cdot)$ is the heat duty of the reboiler. For the implementation of (0.28) in the time domain, we use a state-space representation:

$$z(t)=\mathbf{A}z(t-1)+\mathbf{B}u(t) \tag{0.29a}$$

$$y(t)=\mathbf{C}(p_1,p_2)\,z(t), \tag{0.29b}$$

where the state-space matrices are given by:

$$\mathbf{A}=\begin{bmatrix} 0.99602 & 0 & 0 & 0 \\ 0 & 0.9939 & 0 & 0 \\ 0 & 0 & 0.99683 & 0 \\ 0 & 0 & 0 & 0.99538 \end{bmatrix} \tag{0.30a}$$

$$\mathbf{B}=\begin{bmatrix} 0.066534 & 0 \\ 0.066463 & 0 \\ 0 & 0.066561 \\ 0 & 0.066513 \end{bmatrix} \tag{0.30b}$$

$$\mathbf{C}(p_1,p_2)=\begin{bmatrix} 0.76647+p_1 & 0 & -0.9 & 0 \\ 0 & 0.6055+p_2 & 0 & -1.3472 \end{bmatrix}. \tag{0.30c}$$

The closed-loop error is $\epsilon(t,\xi)=\bar{y}(\xi)-y(t,\xi)$ and the cost function for this problem are:

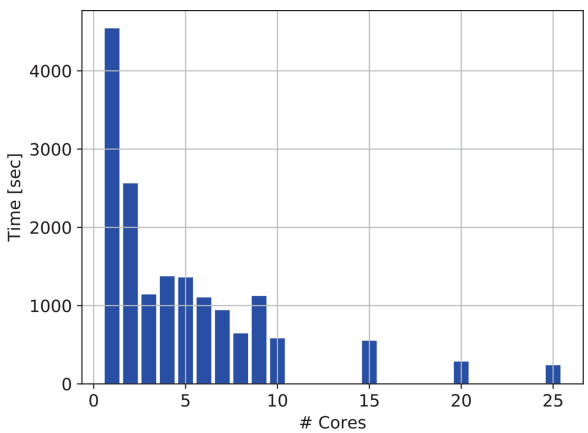

Figure 9. Scalability of PIPS-NLP in distillation tuning problem with $|\Omega|=1000$ scenarios.

[Color figure can be viewed at wileyonlinelibrary.com]

$$\varphi(\epsilon_{\mathcal{T}}, u_{\mathcal{T}}, \xi) = \int_{\mathcal{T}} (\gamma_\epsilon \epsilon(t,\xi)^T \epsilon(t,\xi) + \gamma_u u(t,\xi)^T u(t,\xi)) dt. \tag{0.31}$$

In the SP tuning formulation of this problem, we have four random parameters $\Xi = (\bar{y}_1, \bar{y}_2, p_1, p_2)$. The parameters $p_1 \sim \mathcal{N}(0, 0.1), p_2 \sim \mathcal{N}(0, 0.1)$ capture modeling errors while $\bar{y}_1 \sim \mathcal{U}(-0.05, 0.05)$ and $\bar{y}_2 \sim \mathcal{U}(-0.05, 0.05)$ are output set-points. The problem has six settings $\pi = (K_P^1, K_I^1, K_D^1, K_P^2, K_I^2, K_D^2)$ (three settings for the loop $u_1(t) \rightarrow y_1(t)$ and three settings for loop $u_2(t) \rightarrow y_2(t)$). We seek to tune these settings simultaneously to take into account interactions between the control loops variables. We set the bounds $\bar{K}_c, \bar{K}_I, \bar{K}_D = 100$ and $\bar{u} = 20$ and weights $\gamma_\epsilon = 10$ and $\gamma_u = 0.01$.

We solved this problem with PIPS-NLP. For a problem instance with $|\Omega| = 1000$ scenarios, we obtain an NLP with 1,107,006 variables (six of them are degrees of freedom). This instance can be solved in 23 iterations and 86 s (1.43 min) using 32 computing cores. In Figure 8 we show the closed-loop responses for the two output variables. We can see that both outputs can be brought to the set-point effectively and in essentially all scenarios (despite the presence of variable interactions). However, we see that in one scenario the distillate and bottoms concentration cannot be stabilized. We can thus conclude that the controllers cannot handle the range of uncertainties considered. This highlights how SP techniques can be used to explore the performance limits of control configurations.

In Figure 9 we illustrate the scalability of PIPS-NLP as we increase the number of computing cores for an instance with $|\Omega| = 1000$ scenarios. To obtain these results, we increased the range of the set-point changes. As can be seen, the solution time of this problem on a single computing core is 1.2 h and this time can be brought down to 4.21 min using 20 computing cores (an average of 150 iterations is required in all cases). We also see that solution times quickly decay as we increase the number of cores and scaling is *superlinear* (acceleration is faster than linear). This is because the state-space model creates small dense blocks in the linear system. This problem could not be solved with SNGO because the subproblems are too complex for SCIP. As part of future work, we will investigate alternative strategies to solve the subproblems.

We highlight that the proposed tuning formulation can simultaneously tune multiple controller loops (as in a large chemical plant) because the computational complexity of the problem is similar to those encountered in plant-wide MPC formulations. Moreover, the incorporation of multiple scenarios can be handled by performing Schur decomposition.

Conclusions and Future Work

We have illustrated how to use stochastic programming techniques to perform controller tuning tasks in a systematic manner. We argue that stochastic programming techniques can be used to assess the performance limits of controllers and to shape the cost distribution in desirable ways. We have also illustrated how computing tools can be used to address challenging formulations. With this work we aim to open the door to a number of future directions on controller tuning using stochastic programming techniques. In particular, global optimization is essential in design tasks such as controller tuning because one often wants a certificate that no better controller settings exist. It would be also interesting to explore stochastic programming formulations in the frequency domain, where stability criteria can be easier to enforce. In addition, the proposed techniques can be applied to more advanced control configurations that involve cascades, Smith predictors, and feedforward control capabilities.

Acknowledgments

We thank the Vice Chancellor for Research and Graduate Education at the University of Wisconsin-Madison and the industrial members of the TWCCC consortium for their generous support.

Literature Cited

1. Mayne DQ, Rawlings JB, Rao CV, Scokaert PO. Constrained model predictive control: stability and optimality. *Automatica*. 2000;36(6):789–814.
2. Qin SJ, Badgwell TA. A survey of industrial model predictive control technology. *Control Eng Pract*. 2003;11(7):733–764.
3. Cao Y, D'amato F, Zavala VM. Stochastic optimization formulations for wind turbine power maximization and extreme load mitigation. IEEE Transactions on Energy Conversion, Under Review, 2017.
4. Åström KJ, Hägglund T. The future of PID control. *Control Eng Pract*. 2001;9(11):1163–1175.
5. Åström K, Panagopoulos H, Hägglund T. Design of PI controllers based on non-convex optimization. *Automatica*. 1998;34(5):585–601.
6. Hwang C, Hsiao CY. Solution of a non-convex optimization arising in PI/PID control design. *Automatica*. 2002;38(11):1895–1904.
7. Toscano R. A simple robust PI/PID controller design via numerical optimization approach. *J Process Control*. 2005;15(1):81–88.
8. He JB, Wang QG, Lee TH. PI/PID controller tuning via LQR approach. *Chem Eng Sci*. 2000;55(13):2429–2439.
9. Kim TH, Maruta I, Sugie T. Robust PID controller tuning based on the constrained particle swarm optimization. *Automatica*. 2008;44(4):1104–1110.
10. Morari M, Zafiriou, E. *Robust Process Control*. Englewood Cliffs, NJ: Prentice Hall, 1989.
11. Rivera DE, Morari M, Skogestad S. Internal model control: PID controller design. *Ind Eng Chem Process Des Dev*. 1986;25(1):252–265.
12. Skogestad S. Simple analytic rules for model reduction and PID controller tuning. *Model Identif Control*. 2004;25(2):85–120.
13. Holt BR, Jerome NF, Buck U, Dubinsky M, Economou C, Grimm W, Galimidi A, Grosdidier P, Jordan K, Klewin R, Kraemer W, Kuguenko GV, Mandler J, Ness A, Ninman TA, Plocher T, Rivera DE, Sela R, Szakaly Z, Tung HH, Webb C, Zafiriou E, Morari M, Ray WH. Consydintegrated software for computer aided control system design and analysis. *Comput Chem Eng*. 1987;11(2):187–203.
14. Wang Q-G, Zou B, Lee T-H, Bi Q. Auto-tuning of multivariable pid controllers from decentralized relay feedback. *Automatica*. 1997;33(3):319–330.
15. Åström KJ, Hägglund T, Hang CC, Ho WK. Automatic tuning and adaptation for PID controllers: a survey. *Control Eng Pract*. 1993;1(4):699–714.
16. Ge M, Chiu M-S, Wang Q-G. Robust PID controller design via LMI approach. *J Process Control*. 2002;12(1):3–13.
17. Toscano R. Robust synthesis of a PID controller by uncertain multimodel approach. *Inform Sci*. 2007;177(6):1441–1451.
18. Johnson KE, Pao LY, Balas MJ, Fingersh LJ. Control of variable-speed wind turbines: standard and adaptive techniques for maximizing energy capture. *IEEE Control Syst*. 2006;26(3):70–81.
19. Couchman PD, Cannon M, Kouvaritakis B. Stochastic mpc with inequality stability constraints. *Automatica*. 2006;42(12):2169–2174.
20. Mesbah A. Stochastic model predictive control: an overview and perspectives for future research. *IEEE Control Syst*. 2016;36(6):30–44.
21. Oldewurtel F, Jones CN, Morari M. A tractable approximation of chance constrained stochastic mpc based on affine disturbance feedback. In: *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on IEEE*, 2008:4731–4736.
22. Tøndel P, Johansen TA, Bemporad A. Evaluation of piecewise affine control via binary search tree. *Automatica*. 2003;39(5):945–950.
23. Pflug GC. Some remarks on the value-at-risk and the conditional value-at-risk. In: Uryasev SP, editor. *Probabilistic Constrained Optimization*. Boston, MA: Springer, 2000:272–281.

24. Santos-Rodriguez MM, Flores-Tlacuahuac A, Zavala VM. A stochastic optimization approach for the design of organic fluid mixtures for low-temperature heat recovery. *Appl Energy*. 2017;198:145–159.
25. Dolan ED, Moré JJ. Benchmarking optimization software with performance profiles. *Math Program*. 2002;91(2):201–213.
26. Linderoth J, Shapiro A, Wright S. The empirical behavior of sampling methods for stochastic programming. *Ann Operat Res*. 2006; 142(1):215–241.
27. Chen M, Mehrotra S, Papp D. Scenario generation for stochastic optimization problems via the sparse grid method. *Comput Optim Appl*. 2015;62(3):669–692.
28. Klöppel M, Geletu A, Hoffmann A, Li P. Using sparse-grid methods to improve computation efficiency in solving dynamic nonlinear chance-constrained optimization problems. *Ind Eng Chem Res*. 2011; 50(9):5693–5704.
29. Mehrotra S, Papp D. Generating nested quadrature formulas for general weight functions with known moments. *arXiv preprint arXiv:1203.1554*, 2012.
30. Garcke J. Sparse grids in a nutshell. In: Garcke J, Griebel M, editors. *Sparse Grids and Applications*. Berlin, Heidelberg: Springer, 2012:57–80.
31. Burkardt J. Sparse grid cce. 2014. Available from https://people.sc.fsu.edu/~jburkardt/presentations/sparse_2014_fsu.pdf
32. Åström KJ, Murray, RM. *Feedback Systems: An Introduction for Scientists and Engineers*. NJ: Princeton University Press, 2010.
33. Chiang N, Petra CG, Zavala VM. Structured nonconvex optimization of large-scale energy systems using pips-nlp. In: *Power Systems Computation Conference (PSCC)*. IEEE, 2014:1–7.
34. Kang J, Cao Y, Word DP, Laird CD. An interior-point method for efficient solution of block-structured nlp problems using an implicit schur-complement decomposition. *Comput Chem Eng*. 2014;71:563–573.
35. Zavala VM, Laird CD, Biegler LT. Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chem Eng Sci*. 2008;63(19):4834–4845.
36. Cao Y, Zavala VM. A scalable global optimization algorithm for stochastic nonlinear programs. Under Review, 2017.
37. Vigerske S, Gleixner A. Scip: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optim. Methods Softw*. 2017:1–31. DOI: 10.1080/10556788.2017.1335312.
38. Sahinidis NV. Baron: a general purpose global optimization software package. *J Global Optim*. 1996;8(2):201–205.
39. Singer AB, Barton PI. Global optimization with nonlinear ordinary differential equations. *J Global Optim*. 2006;34(2):159–190.
40. Dunning I, Huchette J, Lubin M. Jump: a modeling language for mathematical optimization. *SIAM Rev*. 2017;59(2):295–320.
41. Cao Y, Fuentes-Cortes LF, Chen S, Zavala VM. Scalable modeling and solution of stochastic multiobjective optimization problems. *Comput Chem Eng*. 2017;99:185–197.
42. JalvingAbhyankar J, Kim S, Hereld KM, Zavala VM. A graph-based computational framework for simulation and optimisation of coupled infrastructure networks. *IET Generat Trans Distribut*. 2017;11(12): 3163–3176.
43. Wächter A, Biegler LT. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math Prog*. 2006;106(1):25–57.
44. Watson J-P, Woodruff DL, Hart WE. Pysp: modeling and solving stochastic programs in python. *Math Prog Comput*. 2012;4(2):109–149.
45. Wood R, Berry M. Terminal composition control of a binary distillation column. *Chem Eng Sci*. 1973;28(9):1707–1717.

Manuscript received Sep. 1, 2017, and revision received Oct. 20, 2017.