The SCFO Real-Time Optimization Solver: Users' Guide (version 0.9.4)

Gene A. Bunin, Grégory François, Dominique Bonvin

Laboratoire d'Automatique, Ecole Polytechnique Fédérale de Lausanne

August 18, 2013

Contents

1 Introduction

1.1 What is Real-Time Optimization?

The term "real-time optimization" (RTO) tends to be employed for different purposes by different researchers/practitioners. In our context, we use the term very generally to denote more specific terms like "measurement-based optimization", "on-line optimization", "iterative set-point optimization", "steady-state optimization", "experimental optimization", "iterative feedback optimization", "run-to-run (batch-to-batch, cycle-to-cycle) optimization", and "black-box optimization". It is important to emphasize, however, that we do not mean "optimization in real-time" and that **this solver is not intended for numerical optimization problems with real-time constraints**.

Specifically, the solver presented here is meant for solving problems of the following form:

$$\begin{array}{rl} \underset{\mathbf{u}}{\text{minimize}} & \phi_p(\mathbf{u}) \\ \text{subject to} & \mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0} \\ & \mathbf{G}(\mathbf{u}) \preceq \mathbf{0} \\ & \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \end{array}, \tag{1}$$

where $\mathbf{u} \in \mathbb{R}^{n_u}$ are the manipulated variables (or "inputs"), $\phi_p : \mathbb{R}^{n_u} \to \mathbb{R}$ is the cost function to be minimized, $\mathbf{G}_p$ and $\mathbf{G}$ are composed of, respectively, n_g uncertain and $\bar{n}_g$ certain inequality constraint functions $g_p, g : \mathbb{R}^{n_u} \to \mathbb{R}$ (i.e., physical/safety/user-defined limitations), and $\mathbf{u}^L$ and $\mathbf{u}^U$ denote the "box" constraints (the lower and upper bounds) on the inputs (e.g., actuator limits).

The subscript p is used to refer to functions that are not known analytically and require an "expensive" experiment so as to be evaluated for a given input $\mathbf{u}$ (in process systems lingo, p stands for "plant"). In many cases, such an experiment will be real in nature and expensive because it requires the operation of certain equipment – e.g., running a pharmaceutical batch, manufacturing and testing of a certain part, or applying a certain operating regime to an automatic process. However, it is also possible that the experiment be numerical and expensive because it requires running a long simulation. By contrast, the absence of p is used to denote that the function is known analytically or can be evaluated inexpensively.

The goal of any good RTO algorithm is to solve Problem (1) to local optimality by finding a set of inputs, $\mathbf{u}^*$, that minimizes $\phi_p(\mathbf{u})$ while satisfying all of the constraints. This is always done by iteratively carrying out experimental realizations according to some RTO update law Γ, with k denoting the iteration counter:

$$\mathbf{u}^*_{k+1} = \Gamma(\mathbf{u}_0, ..., \mathbf{u}_k, \mathbf{y}_0, ..., \mathbf{y}_k), \tag{2}$$

with $\mathbf{y}$ used to denote measured output data and $\mathbf{u}^*_{k+1}$ denoting the set of inputs that the RTO algorithm believes to be "optimal", or at least "better" than the current applied $\mathbf{u}_k$. In practice, Γ can be as simple as a human being (or many) who, based on what is observed and measured, makes a decision on what inputs should be applied next to achieve better performance. Alternatively, Γ may also be a complicated algorithmic law that, without requiring anything from the operator, automatically generates new inputs based on what is measured. As many RTO algorithms make judgments about future inputs based on measurements that are inherently local,

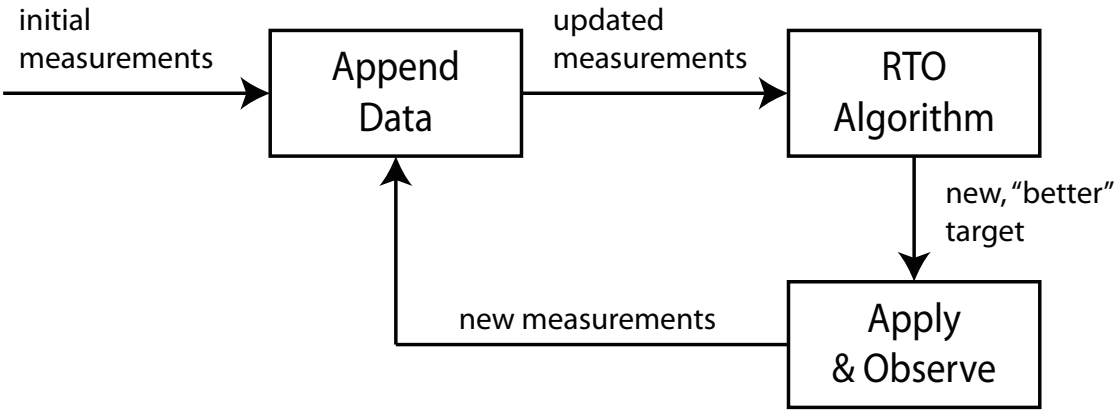

Figure 1: A qualitative illustration of the RTO loop.

setting the next set of inputs as $\mathbf{u}_{k+1} := \mathbf{u}^*_{k+1}$ is not always a wise choice, and an input filter $K_k \in [0, 1]$ may be used instead:

$$\mathbf{u}_{k+1} := \mathbf{u}_k + K_k \left(\mathbf{u}^*_{k+1} - \mathbf{u}_k \right), \tag{3}$$

so as to "dampen" the adaptation.

It is generally desired that the RTO scheme described by (2) and (3) converge without needing too many iterations, as these are usually expensive. Additionally, it may be required that the iterates generated by the algorithm satisfy certain constraints at all iterations, as not doing so may compromise the safety of the personnel or the functionality of the equipment. A simple schematic of the iterative RTO procedure is given in Figure 1.

1.2 What is SCFO and Why Use It?

"SCFO" is the acronym adopted by the authors, and stands for the "sufficient conditions for feasibility and optimality" as proposed by the authors with the hopes of creating a theoretical foundation for RTO theory and analysis [8, 9]. Practically, we will use "SCFO" to refer to the RTO framework designed around these conditions, as well as to the solver presented here, which uses this framework.

The motivation for using the SCFO is that it is, to date, the only tool theoretically capable of guaranteeing that the loop of Figure 1 converge to a local minimum while keeping the safety constraints satisfied. As an additional benefit, the SCFO also ensure that the cost value improves at every iteration, which acts as a conceptual guarantee that the user applying the SCFO can only gain by doing so. The qualitative proof of convergence is simple and based on the following logic:

$$\begin{array}{l} \bullet \text{ every iterate is feasible} \\ \bullet \text{ every iterate is better} \end{array} \Rightarrow \begin{array}{l} \text{convergence when no better feasible} \\ \text{iterate exists (i.e., local minimum)} \end{array},$$

with the theoretical details relegated to [8].

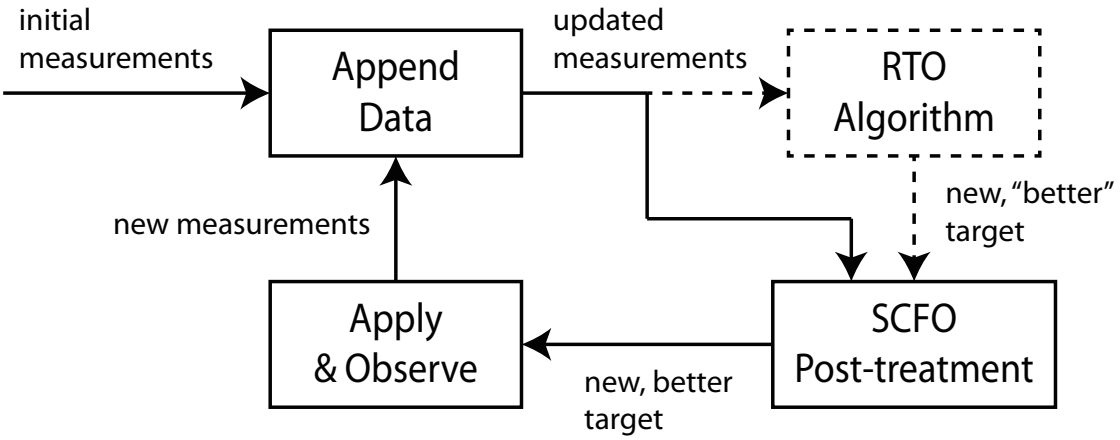

Figure 2: A qualitative illustration of the RTO loop with the SCFO post-treatment included. The dashed lines are used to indicate that the RTO algorithm is not necessary in this case.

Not surprisingly, such desirable theoretical guarantees cannot be preserved in practical applications, as much of the information needed to guarantee that the SCFO works conceptually is either corrupted or unavailable. However, it has been shown that robust extensions of the SCFO are easily formulated and applied [9], thereby allowing for the SCFO framework to retain, albeit approximately, a lot of its strong qualities in more realistic settings. Schematically, it may be seen as a "post-treatment" of the answer provided by a given RTO algorithm, which ensures that the "better" target really is better (Figure 2). We point out that the SCFO framework is purely data-driven – indeed, the essential role of the SCFO solver is to manage the obtained data intelligently so as to force feasible-side convergence to the minimum. Additionally, the RTO algorithm is not a necessity when the SCFO is used, as the SCFO can single-handedly use the obtained data to solve the problem. However, it is still recommended that some intelligent RTO algorithm is used as this aids in the convergence speed of the overall scheme.

1.3 Can I Use This Solver for My Problem?

1.3.1 Key Requirement

To apply the SCFO to your problem, the essential requirement is that you be able to phrase what you want to do in the form of Problem (1). If you simply cannot do this, then the SCFO solver is not applicable for you. If the formulation of Problem (1) seems too abstract, then we urge you to have a look at Section 4, where some examples of applications where RTO problems naturally appear are presented in a tutorial manner, together with detailed explanations of how the SCFO solver could be configured and applied in these scenarios.

1.3.2 Additional Conditions and Caveats

Static problems

If the functions of Problem (1) are transient and depend on time as well as the inputs:

$$\begin{array}{rl} \underset{\mathbf{u}}{\text{minimize}} & \phi_p(\mathbf{u}, t) \\ \text{subject to} & \mathbf{G}_p(\mathbf{u}, t) \preceq \mathbf{0} \\ & \mathbf{G}(\mathbf{u}, t) \preceq \mathbf{0} \\ & \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \end{array},$$

then you may not be able to apply the SCFO to your problem. In some cases, it may be possible to reformulate the dynamic problem as a static one simply by working exclusively with steady-state mappings (i.e., applying a certain $\mathbf{u}$ and waiting for the dynamics of the system to die out so as to remove the dependance on t and return to the original formulation). Users interested in this reformulation are referred to Section 4.1 and the examples therein.

"Reasonable" dimensionality

Ideally, the current version of the SCFO solver should be applied to problems with no more than, as a rough (and ambitious) approximation, 100 manipulated variables. If your problem has thousands or millions of variables that you would like to optimize over, then the current version of the solver is probably not well-suited for you, as the solver's subroutines may not be able to handle your problem size. It is, however, possible that a custom-written modification that takes advantage of certain additional assumptions tailored to your problem may work well.

Note as well that convergence speed will also change with the number of input variables. While the exact trends of how exactly the SCFO scale up are still unknown and may be problem-dependent, it is safe to say that this scale-up will, in the ideal case, be no better than linear. Put otherwise, if a problem with 10 inputs converges in approximately 30 iterations, a 100-input problem of similar structure cannot be expected, on average, to converge in less than 300. As such, just because you have 100 inputs in your RTO problem should not mean that you should try to optimize over all of them – it is recommended, instead, that you choose the subset of the inputs that is the most crucial and optimize over those. Future versions of the solver will look into possible automations of which inputs to optimize over.

Continuous functions

The solver is not theoretically equipped to handle RTO problems with discrete variables or constraints. For example, the problem

$$\begin{array}{rl} \underset{\mathbf{u},z}{\text{minimize}} & \phi_p(\mathbf{u}, z) \\ \text{subject to} & \mathbf{G}_p(\mathbf{u}, z) \preceq \mathbf{0} \\ & \mathbf{G}(\mathbf{u}, z) \preceq \mathbf{0} \\ & \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \\ & z \in \{0, 1\} \end{array}$$

is not rigorously solvable with the current tools. Fundamentally, this is because the SCFO relies very heavily on gradients, which do not exist for the discrete cases, thereby making it impossible to treat the problem in the same rigorous sense as is done for the continuous inputs.

That being said, it is nevertheless the case that *all* real problems are discrete in nature due to finite precision, and can only approximate the continuous case. When the round-off error is negligible (e.g., when all the inputs $\mathbf{u}$ are rounded to, say, the tenth decimal place), the corresponding inputs may be, without serious consequences, treated as continuous. This is,

however, less justifiable when the round-off error is great (e.g., to the nearest integer). As an example, one could treat z as continuous in the problem above, and then simply round whatever z^* value an RTO algorithm provides to either 0 or 1 – this is not, however, theoretically guaranteed to produce good results.

A theoretically rigorous, though more expensive, approach would be to fix the discrete variables and solve the resulting continuous RTO problem for each possible combination (i.e., once for $z = 0$ and once for $z = 1$ in the example above).

Local solutions

The SCFO solver is designed for converging to a *local* minimum in the neighborhood of the best known input point, and is by no means a rigorous global optimization algorithm (the existence of which is largely absent for RTO problems). Users who wish to attempt global optimization with the SCFO should do so by sampling different portions of their input space (either intelligently or haphazardly) and then feed this data to the solver, which will proceed to locally minimize in the region of the best available point.

No limiting real-time constraints

The SCFO solver does not promise any strict bounds on the computation time needed to provide a solution. The "slow" (and original) version of the slower may need from seconds to hours, depending on the number of inputs, the amount of data collected, and the number of p functions that are present. The "fast" version sacrifices some theoretical rigor and will usually be a 100^+-fold faster, taking a few seconds or less for many problems. If real-time constraints are relevant for your problem, we suggest that you use the fast version, but warn that you should try out the solver on a similar dummy problem first so as to have an idea of the average computation time needed for your problem. Future releases will work to optimize the internal routines of both the slow and fast versions as well as to seek a compromise by removing certain "optional" subroutines so as to meet user-specified real-time constraints.

No degradation

By "degradation", we refer to the case where the RTO problem changes over the course of operation (i.e., with respect to iteration k):

$$\begin{array}{rl} \underset{\mathbf{u}}{\text{minimize}} & \phi_{p.k}(\mathbf{u}) \\ \text{subject to} & \mathbf{G}_{p,k}(\mathbf{u}) \preceq \mathbf{0} \\ & \mathbf{G}_k(\mathbf{u}) \preceq \mathbf{0} \\ & \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \end{array},$$

which is a common occurrence in practice due to equipment degradation or changes in the surrounding environment. The current version of the solver is not equipped to handle such problems, though this will be accounted for in the next major version. For the time being, users who are worried about degradation are advised only to put their most recent measured data into the SCFO, as opposed to *all* of the data collected since the beginning of operation, as the older data may no longer be valid for the process in its current degraded state.

Certainty in the inputs

The inputs **u** must be certain – i.e., the user must be sure that the **u** specified really is the **u** that is applied to the system, and not some corrupted version. When the inputs **u** are themselves subject to uncertainty or noise, then the SCFO approach is no longer theoretically rigorous, which may (or may not) make it perform worse in practice. Robustness considerations for noisy inputs are currently underway and should be included in a not-too-distant future version, however.

Absence of gross measurement errors

It is assumed that all of the data has been treated for gross measurement errors and any highly erroneous measurements have been removed. Failure to do so may result in an SCFO realization that is highly influenced by the gross errors and that could potentially converge very slowly. Future versions may attempt to integrate a gross-error filter into the solver.

Knowledge of a feasible point

At least one choice of **u** that satisfies the constraints of Problem (1) with some back-off (safety margin) should be available. The size of these back-offs will depend on user specifications (discussed in more detail in Sections 2 and 3).

2 Getting Started

This section is intended to provide the bare necessities for a user to apply the SCFO solver to their problem. For those interested in the inner workings of the code, we discuss a number of theoretical points in Section 3, but avoid them here as much as possible.

2.1 Installation

The current version of the solver is available in both MATLAB and GNU Octave and requires the following files, all of which should be included in the corresponding archives – `SCFOmatV0_9_4.rar` and `SCFOoctV0_9_4.rar` for the MATLAB and Octave versions, respectively. These are available for download at `http://infoscience.epfl.ch` (simply search for "SCFO"), and should be extracted into a directory where MATLAB/Octave will find them:

- `SCFO.m`/`SCFO_oct.m` (the core file)
- `pengen.m`/`pengen_oct.m` (subroutine for constructing a log-concave approximation of the probability density function, or PDF, of the measurement noise)
- `logccvfit.m`/`logccvfit_oct.m` (routine for calculating a log-concave approximation of histogram data)
- `normaldist.m`/`normaldist_oct.m` (normalization of a log-concave function to obtain a log-concave pdf)
- `penlinrep.m`/`penlinrep_oct.m` (piecewise-linear approximation of a log-likelihood penalty function)
- `randgen.m`/`randgen_oct.m` (random number generator for a given PDF)
- `clusterdata.m`/`clusterdata_oct.m` (binning of input points)
- `gradboundM.m`/`gradboundM_oct.m` (calculation of gradient bounds/estimates)
- `gradboundMfast.m`/`gradboundMfast_oct.m` (calculation of gradient bounds/estimates – fast version)
- `reffind.m`/`reffind_oct.m` (smoothing of noisy data to find best reference point)
- `reffindfast.m`/`reffindfast_oct.m` (smoothing of noisy data to find best reference point – fast version)
- `gradboundM0.m`/`gradboundM0_oct.m` (calculation of gradient bounds/estimates for noise-free case)
- `hood.m`/`hood_oct.m` (sorting of input data with respect to distance from reference point)
- `reguboundM.m`/`reguboundM_oct.m` (bisection algorithm to find local neighborhood of structural validity)
- `reguboundMfast.m`/`reguboundMfast_oct.m` (bisection algorithm to find local neighborhood of structural validity – fast version)
- `reguboundM0.m`/`reguboundM0_oct.m` (bisection algorithm to find local neighborhood of structural validity for noise-free case)

- `gennorm.m`/`gennorm_oct.m` (calculation of residual penalties with respect to a given log-likelihood penalty function)
- `respenal.m`/`respenal_oct.m` (mapping between residual and penalty for a given probability distribution)
- `congenM.m`/`congenM_oct.m` (generation of linear constraints for the bisection algorithm)
- `congenM0.m`/`congenM0_oct.m` (generation of linear constraints for the bisection algorithm for the noise-free case)
- `congenM2.m`/`congenM2_oct.m` (generation of linear constraints for the calculation of gradient bounds)
- `congenM20.m`/`congenM20_oct.m` (generation of linear constraints for the calculation of gradient bounds for the noise-free case)
- `feascheck.m`/`feascheck_oct.m` (routine to check the robust feasibility of a given input point)

For MATLAB users, we note that a portion of the convex optimization subroutines are written in, and solved by, the (free) CVX-SeDuMi/SDPT3 package, which should be downloaded separately (`www.cvxr.com`) and installed where, again, MATLAB can find it. It is not necessary to understand how this package works to use the SCFO solver (one only needs to install it), but interested users are referred to [28, 27, 19]. Future versions will do away with this dependence, however, and will give the user a choice of numerical solvers. The Octave version currently accomplishes the same tasks with the built-in Octave linear/quadratic programming routines, although we hope to expand on this as well in the future. We also note that the current MATLAB version is significantly faster than the corresponding Octave one with respect to computation time.

Apart from acquiring the necessary files, the actual "installation" will inevitably depend on the application that the user is applying the SCFO to, and on what acts as the interface between the MATLAB/Octave solver and the data acquisition system. If the data is not in MATLAB/Octave to begin with, then it is up to the user to export it to MATLAB/Octave, to run the SCFO, and then to import the solution back into the system.

2.2 Running `SCFO.m`/`SCFO_oct.m`

The current version of the solver requires a total of 27 input elements, of which some are **fixed**, some are **easily chosen**, and some **need to be chosen with care**. We explain all of these here and try to place particular emphasis on the latter. The overall user input has the following symbolic form[1]:

$$[\mathbf{u}_{k+1}, \delta_{exit}] = \texttt{SCFO}\left(\begin{array}{c} \mathbf{U}, \hat{\boldsymbol{\phi}}, \hat{\mathbf{G}}_p, \mathbf{u}^*_{k+1}, \mathbf{w}_\phi, \mathbf{W}_g, \mathbf{C}, \mathbf{d}, \mathbf{d}_T, p_\phi, \mathbf{u}^L, \mathbf{u}^U, \underline{\mathbf{K}}_p, \overline{\mathbf{K}}_p, \\ \underline{\mathbf{K}}, \overline{\mathbf{K}}, \underline{\boldsymbol{\kappa}}_\phi, \overline{\boldsymbol{\kappa}}_\phi, \underline{\mathbf{M}}, \overline{\mathbf{M}}, \underline{\mathbf{G}}_p, \underline{\mathbf{G}}, \underline{\phi}, \epsilon_\phi, \Delta\mathbf{u}_{max}, D, \mathcal{F} \end{array}\right),$$

with the notation:

- $\delta_{exit} \in \{0, 1, 2\}$: exit condition
- $\mathbf{U} \in \mathbb{R}^{(k+1)\times n_u}$: input data collected to date

[1] This should be replaced with `SCFO_oct()` for Octave users.

- $\hat{\boldsymbol{\phi}} \in \mathbb{R}^{(k+1)\times 1}$: cost data collected to date
- $\hat{\mathbf{G}}_p \in \mathbb{R}^{(k+1)\times n_g}$: uncertain constraint data collected to date
- $\mathbf{u}^*_{k+1} \in \mathbb{R}^{1\times n_u}$: new input target as provided by RTO algorithm
- $\mathbf{w}_\phi$: noise samples of cost function measurements/estimates
- $\mathbf{W}_g$: noise samples of uncertain constraint function measurements/estimates
- $\mathbf{C} \in \mathbb{Z}^{n_g\times n_u}$: Boolean matrix of concave relationships between uncertain constraints $\mathbf{G}_p$ and inputs $\mathbf{u}$
- $\mathbf{d} \in \mathbb{R}_+^{1\times n_g}$: maximal allowable violations of uncertain constraints
- $\mathbf{d}_T \in \mathbb{R}_+^{1\times n_g}$: maximal allowable violation integrals for the uncertain constraints
- $p_\phi \in \{0, 1\}$: indicator of certainty in the cost function
- $\mathbf{u}^L \in \mathbb{R}^{1\times n_u}$: lower limits on the inputs
- $\mathbf{u}^U \in \mathbb{R}^{1\times n_u}$: upper limits on the inputs
- $\underline{\mathbf{K}}_p \in \mathbb{R}^{n_g\times n_u}$: lower Lipschitz constants on the uncertain constraints
- $\overline{\mathbf{K}}_p \in \mathbb{R}^{n_g\times n_u}$: upper Lipschitz constants on the uncertain constraints
- $\underline{\mathbf{K}} \in \mathbb{R}^{\bar{n}_g\times n_u}$: lower Lipschitz constants on the certain constraints
- $\overline{\mathbf{K}} \in \mathbb{R}^{\bar{n}_g\times n_u}$: upper Lipschitz constants on the certain constraints
- $\underline{\boldsymbol{\kappa}}_\phi \in \mathbb{R}^{1\times n_u}$: lower Lipschitz constants on the cost
- $\overline{\boldsymbol{\kappa}}_\phi \in \mathbb{R}^{1\times n_u}$: upper Lipschitz constants on the cost
- $\underline{\mathbf{M}} \in \mathbb{R}^{n_u\times n_u}$: lower quadratic bound constants on the cost
- $\overline{\mathbf{M}} \in \mathbb{R}^{n_u\times n_u}$: upper quadratic bound constants on the cost
- $\underline{\mathbf{G}}_p \in \mathbb{R}_-^{1\times n_g}$: lower bounds on the uncertain constraint values
- $\underline{\mathbf{G}} \in \mathbb{R}_-^{1\times \bar{n}_g}$: lower bounds on the certain constraint values
- $\underline{\phi} \in \mathbb{R}$: global lower bound on the cost
- $\epsilon_\phi \in \mathbb{R}_+$: tolerance for which convergence to a sufficiently optimal point may be declared
- $\Delta\mathbf{u}_{max} \in \mathbb{R}_{++}^{1\times n_u}$: maximal step sizes for the inputs
- $D \in \{0, 1, 2\}$: display settings of the solver
- $\mathcal{F} \in \{0, 1\}$: slow/fast computation times

So as to make our detailed explanations of these elements a bit more concrete, we use a modification of the RTO problem example from [8]:

$$\begin{array}{ll} \underset{u_1,u_2}{\text{minimize}} & \phi_p(\mathbf{u}) := (u_1 - 0.5)^2 + (u_2 - 0.4)^2 \\ \text{subject to} & g_{p,1}(\mathbf{u}) := -6u_1^2 - 3.5u_1 + u_2 - 0.6 \leq 0 \\ & g_{p,2}(\mathbf{u}) := 2u_1^2 + 0.5u_1 + u_2 - 0.75 \leq 0 \\ & g_1(\mathbf{u}) := -u_1^2 - (u_2 - 0.15)^2 + 0.01 \leq 0 \\ & u_1 \in [-0.5, 0.5], u_2 \in [0, 0.8] \end{array} , \tag{4}$$

and show how the different elements may be set for this particular problem before applying the SCFO solver to solve it. We remind the user that the p functions in (4), though given analytically here for the purposes of illustration, are assumed unknown to the user, and that the only information that may be obtained is that which is measured when a given input is applied.

2.2.1 The Data: $\mathbf{U}, \hat{\boldsymbol{\phi}}, \hat{\mathbf{G}}_p$

This is the measurement/estimation data obtained from previous iterations (experiments) and should include as many samples as possible, provided that they are valid for the problem at hand (i.e., the problem and equipment have not changed since the data has been collected). $\mathbf{U}$ is simply the matrix of previous input sets that have been applied (stacked vertically), with $\hat{\boldsymbol{\phi}}$ and $\hat{\mathbf{G}}_p$ the corresponding values of the cost and uncertain constraints that were measured or estimated when those inputs were applied to the real system. For Problem (4), we can suppose the case where the points $\mathbf{u}_0 = [0, 0], \mathbf{u}_1 = [0.1, 0.1], \mathbf{u}_2 = [-0.3, 0.4]$, and $\mathbf{u}_3 = [0.4, 0.2]$ have been previously applied, which would yield:

$$\mathbf{U} := \begin{bmatrix} 0 & 0 \\ 0.1 & 0.1 \\ -0.3 & 0.4 \\ 0.4 & 0.2 \end{bmatrix}.$$

The corresponding measurements for the cost and constraints would then be:

$$\hat{\boldsymbol{\phi}} := \begin{bmatrix} 0.41 + w_{\phi,0} \\ 0.25 + w_{\phi,1} \\ 0.64 + w_{\phi,2} \\ 0.05 + w_{\phi,3} \end{bmatrix}, \quad \hat{\mathbf{G}}_p := \begin{bmatrix} -0.60 + w_{1,0} & -0.75 + w_{2,0} \\ -0.91 + w_{1,1} & -0.58 + w_{2,1} \\ 0.31 + w_{1,2} & -0.32 + w_{2,2} \\ -2.76 + w_{1,3} & -0.03 + w_{2,3} \end{bmatrix},$$

with w denoting either measurement noise or estimation errors that corrupt the user's perception of the true cost and constraint values ($\hat{\boldsymbol{\phi}}$ and $\hat{\mathbf{G}}_p$ usually simply being constructed from sensor values or the outputs of an estimator). $\hat{\mathbf{G}}_p$ should be defined by the empty brackets `[]` when there are no uncertain constraints in the RTO problem ($n_g = 0$).

It is recommended that the order in which the measurements are vertically stacked be chronological, with older data at the top and the latest at the bottom, for those measurements that correspond to input points dictated by the SCFO solver. Measurements collected outside of SCFO solver operation are recommended to be placed first and in the order of decreasing cost – i.e., with the data point not obtained by the SCFO and having the lowest cost being the last of the first set, right above the set of measurements that are due to SCFO operation. These are only recommendations, however, and are not crucial, though they may improve performance in some problems. It is, however, crucial that the rows be properly aligned between $\mathbf{U}, \hat{\boldsymbol{\phi}}$, and $\hat{\mathbf{G}}_p$ (e.g., the data in the third rows of $\hat{\boldsymbol{\phi}}$ and $\hat{\mathbf{G}}_p$ should correspond to what was obtained when the input in the third row of $\mathbf{U}$ was applied).

It is required that at least one of the data points provided should satisfy the constraints of the RTO problem robustly and with some satisfactory slack. If one is not available, then the SCFO will quit with `Error: Provided data should include at least one strictly feasible point.` See Section 5.1 for ways to avoid this error.

2.2.2 Nominal RTO Target: $\mathbf{u}_{k+1}^*$

As shown in Figure 2, the SCFO may (and should) be coupled with an RTO algorithm, which provides the solver with a nominal optimal input target $\mathbf{u}_{k+1}^*$. If this is the case, then simply input it in this field as a row vector. If you do not want to use an RTO algorithm and want the SCFO to optimize everything for you, then simply enter the empty brackets `[]`.

2.2.3 Noise Samples: $\mathbf{w}_\phi, \mathbf{W}_g$

$\mathbf{w}_\phi$ represents any available samples of the noise (estimation error) that corrupts the observed cost measurements (estimations):

$$\hat{\phi}_p(\mathbf{u}_k) = \phi_p(\mathbf{u}_k) + w_{\phi,k}.$$

As it will usually be impossible to determine $w_{\phi,k}$ online during optimization, the user is required to have obtained these noise samples beforehand. It is important to highlight that `SCFO.m`/ `SCFO_oct.m` does not accept noise models directly (e.g., one cannot specify that $\mathbf{w}_\phi$ comes from a normal distribution with a certain mean and standard deviation), and that one is obliged to provide a sufficient number (100+) of noise samples instead. To specify a particular model, we recommend that the user supply random samples generated from the model of the distribution, with 10^6 samples being a good amount. For example, to specify that the noise in the cost is Gaussian with a mean of 0 and a variance of 1, one may input the MATLAB/Octave command `randn(1,1e6)` to define $\mathbf{w}_\phi$. If the noise is negligible, the user is encouraged to input the empty brackets `[]`, as this will use subroutines that are, in general, faster and more reliable.

The reason for specifying the noise model this way (as opposed to specifying the nature and parameters of the distribution function) is flexibility – any noise distribution may be specified by the user without the need for explicit parameterizations, with the user also able to input collected noise samples directly (without having to fit a particular model to them). Future versions of the solver will incorporate some standard noise parameterizations, however. The current version also assumes that the noise is independent from sample to sample and that the noise for each measurement/estimate comes from the same probability distribution.

Likewise, $\mathbf{W}_g$ corresponds to a MATLAB/Octave `cell` that contains the noise samples for the different uncertain constraints, i.e., the w elements of

$$\begin{matrix} \hat{g}_{p,1}(\mathbf{u}_k) = g_{p,1}(\mathbf{u}_k) + w_{1,k} \\ \hat{g}_{p,2}(\mathbf{u}_k) = g_{p,2}(\mathbf{u}_k) + w_{2,k} \end{matrix}.$$

Below is an example of how the noise parameters may be inputted for Problem (4) when $w_\phi \sim \mathcal{N}(0, 0.64)$, w_1 is negligible, and $w_2 \sim \mathcal{U}(0, 0.3)$:

$\mathbf{w}_\phi \rightarrow$ `wphi = 0.8*randn(1,1e6);`
$\mathbf{W}_g \rightarrow$ `Wg{1} = []; Wg{2} = 0.3*rand(1,1e6);`

$\mathbf{W}_g$ should be defined by the empty brackets `[]` if there are no uncertain inequality constraints in the RTO problem ($n_g = 0$).

2.2.4 Concavity Matrix: C

In certain applications, relationships between the uncertain constraints and the different input variables may be *concave*, which may lead to significant speed-ups in convergence speed (users interested in an introduction to some concave functions are advised to check the examples section of [13] or Chapter 3 of [1], and to keep in mind that the negative of a convex function is concave – note, as well, that any linear relationship is concave). The matrix $\mathbf{C}$ consists of zeros and ones denoting the absence or the presence, respectively, of a concave relationship between a given constraint and an input. In Problem (4), one could define $\mathbf{C}$ as

$$\mathbf{C} := \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},$$

since $g_{p,1}$ is concave in both u_1 and u_2 (hence, the first row of $\mathbf{C}$), and $g_{p,2}$ is only concave (linear) in u_2 (the second row). It is important to note that this knowledge may, in many cases, be unavailable, in which case we strongly advise the user to set 0 everywhere where they are not confident of a concave relationship existing, since specifying such a relationship when one cannot be verified may lead to hazardous RTO operation. If no concave/linear relationships can be verified for the process being optimized, then the user should define $\mathbf{C}$ as the $\mathbf{0}$ matrix. If there are no uncertain inequality constraints in the RTO problem ($n_g = 0$), then $\mathbf{C}$ should be defined by the empty brackets `[]`.

2.2.5 Constraint Relaxations: $\mathbf{d}, \mathbf{d}_T$

Choosing $\mathbf{d} := \mathbf{0}$ will, in theory, ensure that the uncertain inequality constraints $\mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0}$ are always satisfied at every RTO iteration. This can, however, lead to slow convergence in some cases due to conservatism. By setting the elements of $\mathbf{d}$ to values greater than 0, one allows for certain maximal violations during convergence, which tends to speed up RTO progress significantly in many cases. For example, by specifying

$$\mathbf{d} := [1 \;\; 2]$$

for Problem (4), one is stating that violations of up to $g_{p,1}(\mathbf{u}) = 1$ or $g_{p,2}(\mathbf{u}) = 2$ are tolerable *sometimes*, provided that they do not last forever. To ensure that the violations are temporary, the user must also specify the violation "integrals" $\mathbf{d}_T$, where a violation integral for a given constraint denotes the total sum of violation magnitudes allowed during all RTO operation. For example, specifying

$$\mathbf{d}_T := [10 \;\; 10]$$

for the same problem limits the occurrence of $g_{p,1}(\mathbf{u}) = 1$ and $g_{p,2}(\mathbf{u}) = 2$ to 10 and 5 iterations, respectively. Note that $\mathbf{d} \preceq \mathbf{d}_T$ *always*. Note as well that allowing violations does not guarantee that they will occur, or that the worst-case violation magnitudes will come close to being observed. As such, we strongly recommend the user to exploit this option whenever possible.

2.2.6 Cost Certainty Indicator: p_ϕ

In some applications the cost function will be uncertain (and denoted with a p), in which case p_ϕ should be set to 0. However, significant speed-ups in computation, convergence, and performance will be attained if your RTO problem happens to have an analytically *known* cost function, thereby yielding the RTO problem:

$$\begin{array}{rl} \underset{\mathbf{u}}{\text{minimize}} & \phi(\mathbf{u}) \\ \text{subject to} & \mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0} \\ & \mathbf{G}(\mathbf{u}) \preceq \mathbf{0} \\ & \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \end{array}.$$

In this case, set $p_\phi := 1$ and specify an auxiliary function, `phieval.m`, that calculates the value of the cost function and its derivatives for a given input point. Suppose that the cost function is known for Problem (4). A suitable auxiliary file would then be:

```
function [phi,Jphi] = phieval(u)
phi = (u(1)-.5)^2 + (u(2)-.4)^2;
Jphi = [2*(u(1)-.5) 2*(u(2)-.4)];
end
```

Note that the gradient should be outputted in row format.

2.2.7 Input Bounds: $\mathbf{u}^L, \mathbf{u}^U$

The input bounds should simply define the input space that the user has chosen to work with. For Problem (4), they would be:

$$\mathbf{u}^L := [-0.5 \;\; 0], \;\; \mathbf{u}^U := [0.5 \;\; 0.8].$$

Though this is not obligatory, we *highly recommend* the user to scale the inputs of their RTO problem so that:

$$u_1^U - u_1^L \approx u_2^U - u_2^L \approx ... \approx u_{n_u}^U - u_{n_u}^L \approx 1. \tag{5}$$

2.2.8 Lipschitz Constants: $\mathbf{K}, \boldsymbol{\kappa}$

The Lipschitz constants – $\underline{\mathbf{K}}_p, \overline{\mathbf{K}}_p, \underline{\mathbf{K}}, \overline{\mathbf{K}}, \underline{\boldsymbol{\kappa}}_\phi, \overline{\boldsymbol{\kappa}}_\phi$ – act as lower and upper bounds on the *sensitivities* of the functions involved in the RTO problem and play a very crucial role in the SCFO solver. For the uncertain inequality constraints, the matrices $\underline{\mathbf{K}}_p$ and $\overline{\mathbf{K}}_p$ consist of the Lipschitz constants $\underline{\kappa}_{ji}$ and $\overline{\kappa}_{ji}$ that are defined as:

$$\underline{\kappa}_{ji}^p < \left.\frac{\partial g_{p,j}}{\partial u_i}\right|_{\mathbf{u}} < \overline{\kappa}_{ji}^p, \;\; \forall \mathbf{u} \in \{\mathbf{u} : \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U\}. \tag{6}$$

We may work these constants out for Problem (4) by noting that the derivatives are:

$$\left.\frac{\partial g_{p,1}}{\partial u_1}\right|_{\mathbf{u}} = -12u_1 - 3.5, \quad \left.\frac{\partial g_{p,1}}{\partial u_2}\right|_{\mathbf{u}} = 1$$
$$\left.\frac{\partial g_{p,2}}{\partial u_1}\right|_{\mathbf{u}} = 4u_1 + 0.5, \quad \left.\frac{\partial g_{p,2}}{\partial u_2}\right|_{\mathbf{u}} = 1 \qquad ,$$

and that minimizing and maximizing these derivatives over $u_1 \in [-0.5, 0.5], u_2 \in [0, 0.8]$ leads to the Lipschitz matrices:

$$\underline{\mathbf{K}}_p := \begin{bmatrix} -9.51 & 0.99 \\ -1.51 & 0.99 \end{bmatrix}, \quad \overline{\mathbf{K}}_p := \begin{bmatrix} 2.51 & 1.01 \\ 2.51 & 1.01 \end{bmatrix},$$

where a slack of 0.01 has been added to ensure strict inequality.

Because the analytical expressions for the uncertain plant constraints will never be available, this method of calculating the Lipschitz constants is purely academic and of little practical interest. Unfortunately, there is no "easy", general way to choose these constants for the time being, and we thus recommend the user to adopt an approach that is specific to their problem, where certain physical laws or operation experience may be used to put bounds on the derivatives. For RTO applications that use a model, however, one could calculate the constants as is done above *for the model* and then, for example, to multiply these values by some safety factor (see Section 4.1.2 for an example). Heuristic data-driven approaches are also possible (see the example in Section 4.3.2), and some theoretical data-driven methods are also being developed, though these are not yet integrated into the current version.

Choosing $\underline{\mathbf{K}}_p$ or $\overline{\mathbf{K}}_p$ whose elements are not sufficiently low or high, respectively, can have significant negative effects on RTO performance – most notably, not defining proper Lipschitz bounds can lead to safety constraints $\mathbf{G}_p$ being violated during RTO operation. As such, we advice the user to err on the side of caution and make conservative choices if necessary, even though making the choices too conservative could have adverse effects on performance with respect to convergence speed. In the case where the RTO problem has no uncertain constraints, the user should input the empty brackets `[]` for both $\underline{\mathbf{K}}_p$ and $\overline{\mathbf{K}}_p$. We do note that the solver has a built-in consistency check to ensure that the basic Lipschitz lower and upper bounds are satisfied by the collected measurements (Section 3.10), with $\underline{\mathbf{K}}_p$ and $\overline{\mathbf{K}}_p$ automatically made more conservative in the case that the user-specified constants are found to be inconsistent.

The choices of $\underline{\mathbf{K}}$ and $\overline{\mathbf{K}}$ are analogous but are for the constraints that are analytically known. In this case, the method of calculating the Lipschitz constants by analytically writing out the derivatives and minimizing/maximizing these values over the relevant input space *may* be used. For Problem (4), this would be done as:

$$\left.\frac{\partial g_1}{\partial u_1}\right|_{\mathbf{u}} = -2u_1, \quad \left.\frac{\partial g_1}{\partial u_2}\right|_{\mathbf{u}} = -2u_2 + 0.3,$$

which would lead to:

$$\underline{\mathbf{K}} := [-1.01 \quad -1.31], \quad \overline{\mathbf{K}} := [1.01 \quad 0.31].$$

As for the uncertain case, the user should input the empty brackets `[]` for both $\underline{\mathbf{K}}$ and $\overline{\mathbf{K}}$ if there are no known inequality constraints in the RTO problem ($\bar{n}_g = 0$).

Finally, the Lipschitz constants for the cost – $\underline{\boldsymbol{\kappa}}_\phi$ and $\overline{\boldsymbol{\kappa}}_\phi$ – should be chosen with the same considerations as for the constraints, depending on whether the cost function is known or not. Going again down the analytical path yields:

$$\frac{\partial \phi_p}{\partial u_1}\Big|_{\mathbf{u}} = 2u_1 - 1, \;\; \frac{\partial \phi_p}{\partial u_2}\Big|_{\mathbf{u}} = 2u_2 - 0.8,$$

and

$$\underline{\boldsymbol{\kappa}}_\phi := [-2.01 \;\; -0.81], \;\; \overline{\boldsymbol{\kappa}}_\phi := [0.01 \;\; 0.81]$$

as a suitable choice. When the cost function is unknown, a consistency check is carried out for $\underline{\boldsymbol{\kappa}}_\phi$ and $\overline{\boldsymbol{\kappa}}_\phi$ as well.

2.2.9 The Quadratic Bound Constants: M

The quadratic bound constants $\underline{\mathbf{M}}$ and $\overline{\mathbf{M}}$ are second-order Lipschitz constants that bound the second derivatives of the cost:

$$\underline{M}_{ij} < \frac{\partial^2 \phi_p}{\partial u_i \partial u_j}\Big|_{\mathbf{u}} < \overline{M}_{ij}, \;\; \forall \mathbf{u} \in \{\mathbf{u} : \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U\}.$$

As with the first-order Lipschitz constants, this analytical characterization is of little practical use unless some sort of model of ϕ_p is available, in which case one could attempt to apply the above for the model and then scale these quantities by a safety factor (see example in Section 4.1.2). For Problem (4), we may derive sufficient bounds by noting that:

$$\begin{array}{l} \dfrac{\partial^2 \phi_p}{\partial u_1^2}\Big|_{\mathbf{u}} = 2, \;\; \dfrac{\partial^2 \phi_p}{\partial u_1 \partial u_2}\Big|_{\mathbf{u}} = 0 \\ \dfrac{\partial^2 \phi_p}{\partial u_2 \partial u_1}\Big|_{\mathbf{u}} = 0, \;\; \dfrac{\partial^2 \phi_p}{\partial u_2^2}\Big|_{\mathbf{u}} = 2 \end{array} \rightarrow \begin{array}{l} \underline{M}_{11} := 1.99, \underline{M}_{12} = \underline{M}_{21} := -0.01, \underline{M}_{22} := 1.99 \\ \overline{M}_{11} := 2.01, \overline{M}_{12} = \overline{M}_{21} := 0.01, \overline{M}_{22} := 2.01 \end{array}.$$

Just like with the first-order Lipschitz constants, we cannot claim to have an easy, general approach to determine the quadratic bounds for any given RTO problem, and urge the user to employ whatever *a priori* knowledge is at their disposal to choose them wisely. As with the first-order case, some heuristic data-driven methods may be employed and have been known to work, however (Section 4.3.2).

There are also some cases where this choice is either straightforward or unnecessary. If the cost is linear in all of the variables, then both $\underline{\mathbf{M}}$ and $\overline{\mathbf{M}}$ may be set to approximately (with slack) $\mathbf{0}$. In the case where the cost is known ($p_\phi = 1$), $\underline{\mathbf{M}}$ and $\overline{\mathbf{M}}$ are not needed and the user may input the empty brackets `[]` for both.

2.2.10 Lower Scaling Bounds

The bounds $\underline{\mathbf{G}}_p$ and $\underline{\mathbf{G}}$ are used to scale the constraint values for the subroutines used in the solver. Very simply, $\underline{\mathbf{G}}_p$ denotes the lowest values that the uncertain constraints can take during operation. For Problem (4), a suitable choice would be:

$$\underline{\mathbf{G}}_p := [-3.85 \quad -1].$$

Likewise, $\underline{\mathbf{G}}$ would be the same but for the known constraints. For our example problem, $\underline{\mathbf{G}} := -0.67$ would be sufficient. The empty brackets `[]` should be inputted for the case where either $n_g = 0$ or $\bar{n}_g = 0$ (or both).

We do not expect these choices to be difficult, although this may vary from problem to problem. It should also be noted that these choices are not required to be rigorous/robust and may be approximate.

2.2.11 Global Minimum Cost Value and Tolerance

In most applications, the user should have a reasonably good idea of the best cost function value (the globally minimal value) that they would like to achieve, and should define this value as $\underline{\phi}$. Additionally, it will often be the case that coming within a certain tolerance of this value is sufficient, and that from a practical point of view one does not need to actually reach $\underline{\phi}$ but is satisfied converging close to it. This tolerance, defined by ϵ_ϕ, is a positive number that should reflect the degree of precision with which one would like to converge. Once the solver is provided with a data point, $\mathbf{u}_k$, that robustly satisfies the constraints and the condition

$$\phi_p(\mathbf{u}_k) \leq \underline{\phi} + \epsilon_\phi,$$

no adaptation is done as this point is already known to be sufficiently optimal. This is of practical interest since the solver may make performance temporarily worse due to its continuous exploration of the input space. This exploration, though beneficial for optimization and ultimately necessary in the long run, is no longer so if optimal conditions have already been found.

2.2.12 Maximum Input Step Size: $\Delta\mathbf{u}_{max}$

In principle, one does not need to limit the step size in the solver since everything in the SCFO framework is designed in such a way so as to have desired step sizes implemented automatically. However, while the solver does everything it can with the knowledge it has to drive the process towards truly better performance, multiple errors (e.g., poor user specifications, significant noise corruption in the data, low quality of gradient estimates) could create scenarios where the SCFO may adapt the inputs in an undesirable manner. In this case, so as not to adapt the inputs too much (thereby potentially causing significant losses), we allow the user to put upper limits on the steps that the adaptation may take for the different inputs. These choices are largely *ad hoc* and will depend on the problem and user preferences, but some general-purpose heuristics could be proposed. For example, one might choose to define the maximum step for an input as 5% or 10% of the total domain size of that input. For Problem (4), a 10% limit would lead to:

$$\Delta\mathbf{u}_{max} := [0.10 \;\; 0.08].$$

2.2.13 Solver Display Options: D

The display options do not affect solver performance in any way and are only there to allow the user to control how much output they would like to see from the solver during a single set of computations. Users who are not at all interested in what the solver is actually doing and only care about the answer that it outputs are advised to set $D := 0$, which results in nothing being displayed. Those who want the main steps to be displayed as they are accomplished – perhaps so as to have an idea of how close the solver is to outputting an answer – but do not want all the details are advised to set $D := 1$. Setting $D := 2$ leads to a detailed display that

- Informs the user of the progress of the gradient estimation subroutines (i.e., for which function the gradient is being estimated and how many points are being used for the regularized estimate – see Section 3.5), as well as of the progress of the data smoothing routine to compute the best reference point (see Section 3.9).
- Informs the user if certain Lipschitz constants have been found inconsistent and have thus been automatically modified by the solver – see Section 3.10. If the constants have been modified, both the user-specified and the new choices are displayed.

The detailed display may be of use to those who want to debug/test certain parts of the solver themselves.

2.2.14 Computation Time Option: $\mathcal{F}$

If the time needed for the solver to provide a solution is not a limiting factor, then we suggest setting $\mathcal{F} := 0$, which leads to the standard subroutines being employed. From a theoretical point of view, these are more "proper" and should be employed if permitted. However, if the standard computation time is too slow and there is a significant need/interest in speeding up the solver, the user may set $\mathcal{F} := 1$, which activates the fast version of the solver. Contrary to the standard version, the rigor of certain subroutines (see Section 3.5) is sacrificed in favor of speed and a 100-fold or greater improvement in solution time is usually obtained. In our limited experience, the performances obtained by the two versions appear comparable for many problems, and so the user should not hesitate to make use of this option.

2.2.15 Auxiliary Function for Known Constraints

As with the cost when $p_\phi = 1$, an auxiliary function, `geval.m`, should be provided to evaluate the known constraints ($\mathbf{G}$) and their gradients as a function of $\mathbf{u}$ when these constraints are present. For the example of Problem (4), this file would appear as follows:

```
function [g,Jg] = geval(u)
g(1,1) = -u(1)^2 - (u(2)-.15)^2 + .01;
Jg = [-2*u(1) -2*(u(2)-.15)];
end
```

Suppose now that we had a modified example where all three constraints were known with certainty:

$$\begin{array}{rl} \underset{u_1,u_2}{\text{minimize}} & \phi_p(\mathbf{u}) := (u_1 - 0.5)^2 + (u_2 - 0.4)^2 \\ \text{subject to} & g_1(\mathbf{u}) := -6u_1^2 - 3.5u_1 + u_2 - 0.6 \leq 0 \\ & g_2(\mathbf{u}) := 2u_1^2 + 0.5u_1 + u_2 - 0.75 \leq 0 \\ & g_3(\mathbf{u}) := -u_1^2 - (u_2 - 0.15)^2 + 0.01 \leq 0 \\ & u_1 \in [-0.5, 0.5], u_2 \in [0, 0.8] \end{array} ,$$

The auxiliary function for this example would appear as follows:

```
function [g,Jg] = geval(u)
g(1,1) = -6*u(1)^2 - 3.5*u(1) + u(2) - .6;
g(1,2) = 2*u(1)^2 + .5*u(1) + u(2) - .75;
g(1,3) = -u(1)^2 - (u(2)-.15)^2 + .01;
Jg = [-12*u(1)-3.5 1;
      4*u(1)+.5 1;
      -2*u(1) -2*(u(2)-.15)];
end
```

2.2.16 Exit Condition: δ_{exit}

In the majority of cases when the solver successfully applies the SCFO, the exit condition $\delta_{exit} = 0$ is outputted. Alternatively, it may occur that the solver is forced to override applying the SCFO conditions in favor of enforcing sufficient excitation (see Section 3.6), in which case $\delta_{exit} = 1$ is outputted. If no adaptation is done since sufficiently optimal conditions have already been found, the solver outputs $\delta_{exit} = 2$.

2.3 Example Problem

We now apply the SCFO solver to Problem (4) with the following specifications:

- An initial input data set of $\mathbf{u}_0 = [-0.45\ 0.05]$, $\mathbf{u}_1 = [-0.44\ 0.05]$, and $\mathbf{u}_2 = [-0.45\ 0.06]$, together with the corresponding (noisy) constraint and cost measurements, is provided.
- A simple gradient-descent RTO algorithm (with a diminishing step size) is used to provide an optimal target, with $\mathbf{u}^*_{k+1} = \mathbf{u}_k - \frac{1}{k}\nabla\phi_p(\mathbf{u}_k)$. Here, we assume, for convenience, to know $\nabla\phi_p(\mathbf{u}_k)$, although any corrupted estimate of the gradient could just as easily be used (employing, e.g., the routine of Section 3.5.3).
- The cost measurements are corrupted with white Gaussian noise of zero mean and a variance of 0.0025. Measurements for $g_{p,1}$ are noise-free as the noise is considered negligible. Measurements for $g_{p,2}$ are corrupted with a noise that is uniformly distributed between -0.05 and 0.05.
- It is assumed that $g_{p,1}$ is concave with respect to u_1 only.
- Maximum violations of 1 and 2 are allowed for $g_{p,1}$ and $g_{p,2}$, respectively, with maximal violation integrals of 10 for both.
- The cost is assumed unknown ($p_\phi := 0$).

- The Lipschitz constants as defined previously are used, but conservatism is added to these values for all of the unknown p functions by multiplying positive upper bounds (or negative lower bounds) by 2 and dividing negative upper bounds (or positive lower bounds) by 2.
- The quadratic bounds for the cost are taken as derived before, but conservatism is added by doubling $\overline{\mathbf{M}}$ and setting $\underline{\mathbf{M}} := \mathbf{0}$.
- The scaling bounds as proposed above are used.
- A global minimum cost value of 0 is chosen. However, converging to within 0.1 of this value is deemed as sufficiently optimal.
- The 10%-limit maximum step sizes are used.
- We use the most detailed level of display, with $D := 2$.
- We do not use the fast computation option and set $\mathcal{F} := 0$.

Taking all of these specifications into account allows us to simulate a hundred iterations of the RTO problem with the following snippet of code[2]:

[2]Reminder: Octave users should use `SCFO_oct()` in place of `SCFO()`.

```
wphi = .05*randn(1,1e6);
Wg{1} = [];
Wg{2} = .05 - .1*rand(1,1e6);
C = [1 0; 0 0];
d = [1 2];
dT = [10 10];
phicert = 0;
uL = [-.5 0];
uU = [.5 .8];
Kmin = [-9.51*2 .5*.99; -1.51*2 .5*.99];
Kmax = [2.51*2 1.01*2; 2.51*2 1.01*2];
Kmincert = [-1.01 -1.31];
Kmaxcert = [1.01 .31];
Kcostmin = [-2.01*2 -.81*2];
Kcostmax = [0.01*2 .81*2];
Mmin = zeros(2,2);
Mmax = 2*[2.01 .01; .01 2.02];
Gmin = [-3.85 -1];
Gmincert = -.67;
phimin = 0;
costtol = .1;
Dumax = [.1 .08];
D = 2;
fast = 0;
u(1,:)  = [-.45 .05];
for i = 1:100
  phi(i,1) = (u(i,1) - .5)^2 + (u(i,2) - .4)^2 + .05*randn;
  G(i,1) = -6*u(i,1)^2 - 3.5*u(i,1) + u(i,2) - .6;
  G(i,2) = 2*u(i,1)^2 + .5*u(i,1) + u(i,2) - .75 + (.1-.2*rand);
  Jphi = [2*(u(i,1)-.5) 2*(u(i,2)-.4)];
  ustar = u(i,:)  - (1/i)*Jphi;
  if i == 1
    u(i+1,:)  = [-.4 .05];
  elseif i == 2
    u(i+1,:)  = [-.45 .09];
  else
    [u(i+1,:),delta_exit] = SCFO(u,phi,G,ustar,wphi,Wg,C,d,dT,phicert,...
                                 ...uL,uU,Kmin,Kmax,Kmincert,Kmaxcert,...
                                 ...Kcostmin,Kcostmax,Mmin,Mmax,Gmin,Gmincert,...
                                 ...phimin,costtol,Dumax,D,fast);
  end
end
```

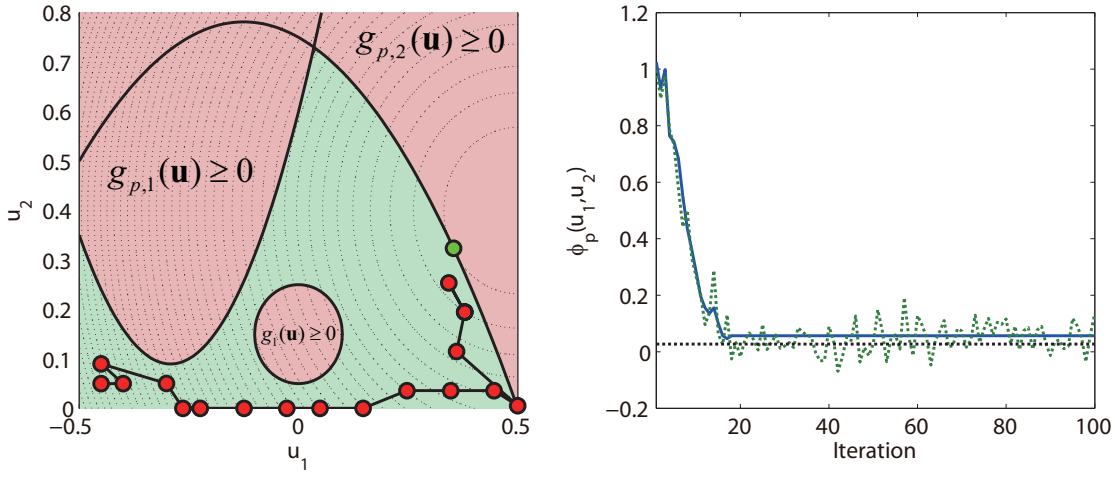

Figure 3: The performance of the SCFO solver for the example problem. The figure on the left shows the input space of the problem, with the red denoting the infeasible regions and the green the feasible ones. Here, the red circles denote the RTO iterations while the green circle denotes the true (unknown) optimum of the RTO problem. The figure on the right shows the value of the cost function, with blue denoting the true (unknown) cost, green denoting the measured, noisy cost, and black denoting the cost at the true optimum.

The results of this simulation may be visualized graphically as shown in Figure 3, where we can see that the solver reaches the neighborhood of the optimum in less 20 iterations and converges there, having found a sufficiently optimal point that satisfies the constraints.

2.4 Some General Guidelines

Below are some basic "rules" that we encourage the user to follow when configuring the SCFO solver. While you may break some of these and still get good performance, it is our belief that these guidelines should be followed in most cases.

2.4.1 Scale Everything

The current version of the solver only does partial scaling (via $\underline{\mathbf{G}}_p$ and $\underline{\mathbf{G}}$) for the subroutines, and it is strongly recommended that the user scale the inputs and the cost/constraint functions to account for major discrepancies. The recommended scaling for the inputs is that of (5). For the cost/constraints, scaling is less crucial but should be done so as to avoid potential numerical difficulties – e.g., if ϕ_p takes values of up to 10^{20}, it is advised to scale it down by dividing by 10^{20}.

2.4.2 Make Bounds as Tight as Possible

The general rule for choosing bounds (e.g., the Lipschitz constants, the quadratic bound constants, the scaling bounds, the input bounds) is to make them as tight as possible without

endangering their validity. For example, suppose that you really are not sure of what the maximal value of $\frac{\partial \phi_p}{\partial u_1}$ is, and so you set $\overline{\kappa}_{\phi,1}$ to a value of 10^5 as something that is very conservative and guaranteed to be valid. When doing so, ask yourself if, for example, a value of 10 is not already sufficient. The latter would be greatly preferred for a number of numerical and performance reasons, provided that it is valid. Similarly, if a given constraint has never been known to go below, for example, -5, then there is probably no reason to set its $\underline{\mathbf{G}}_p$ value to -100, as this could also lead to significant worsening in performance. We note once again, however, that slight invalidities in $\underline{\mathbf{G}}_p$ and $\underline{\mathbf{G}}$ are not expected to have very debilitating effects.

2.4.3 When in Doubt, Err on the Side of Conservatism

Solving RTO problems is not an easy business, and in some applications, it can even be a dangerous business when safety constraints are involved. As such, avoid cheating with assumptions that you cannot back up. Some examples would include setting very tight Lipschitz bounds when they cannot be justified, filling the concavity matrix with 1's even though you are not sure if those concave relationships exist, specifying non-negligible noise as negligible, or treating a p function as certain because you believe that your model "is very good". By contrast, we would advise that you make the Lipschitz constants a bit more conservative than you would like, do not assume concavity unless it is rigorously justified, specify noise models even when there is little noise (and with variances that are slightly greater than the suspected true variances), and do not assume that certain functions are perfectly modeled unless the particular relation is so well-documented and known that there is no reason to assume otherwise. Risky assumptions can sometimes lead to performance gains, but when they fail, they may do so in a very unfavorable manner.

2.4.4 Have an Override Switch Ready

Unlike numerical optimization solvers, where a numerical error or a bug will simply cause the solver to quit with an error message on your computer screen, the SCFO solver acts to dictate real experiments. As such, if you choose to apply the SCFO in a loop in a real system, have some override ready for the unlikely case that `SCFO.m`/`SCFO_oct.m` simply does not return a solution or crashes (for whatever reason). While we have tested, and continue to test, so as to minimize and eliminate this behavior, one should never be too careful when it comes to real experiments.

3 Theoretical Foundations

This section is intended for those interested in the theory and justifications behind the SCFO solver, thereby making it possible for them to look through, and possibly modify, the source code, as well as to make connections with the previously reported [8, 9, 5] – and soon to be reported [7] – material. On the whole, the presentation in this section will still be somewhat cursory (we refer the reader to the cited work for the complete story and mathematical rigor), although a bit more attention will occasionally be given to those subroutines for which the theory has not been previously reported.

3.1 The Theoretical Backbone

The entire SCFO framework is centered around satisfying the following conditions at every iteration k:

$$g_{p,j}(\mathbf{u}_k) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i}), \overline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i})\right) \leq 0, \;\; \forall j = 1, ..., n_g \tag{7}$$

$$g_j(\mathbf{u}_{k+1}) \leq 0, \;\; \forall j = 1, ..., \bar{n}_g \tag{8}$$

$$\mathbf{u}^L \preceq \mathbf{u}_{k+1} \preceq \mathbf{u}^U \tag{9}$$

$$\nabla g_{p,j}(\mathbf{u}_k)^T(\mathbf{u}_{k+1} - \mathbf{u}_k) < 0, \;\; \forall j : g_{p,j}(\mathbf{u}_k) \approx 0 \tag{10}$$

$$\nabla g_j(\mathbf{u}_k)^T(\mathbf{u}_{k+1} - \mathbf{u}_k) < 0, \;\; \forall j : g_j(\mathbf{u}_k) \approx 0 \tag{11}$$

$$\nabla \phi_p(\mathbf{u}_k)^T(\mathbf{u}_{k+1} - \mathbf{u}_k) < 0 \tag{12}$$

$$\nabla \phi_p(\mathbf{u}_k)^T(\mathbf{u}_{k+1} - \mathbf{u}_k) + \frac{1}{2}\sum_{i=1}^{n_u}\sum_{j=1}^{n_u} \max\left(\begin{array}{c} \underline{M}_{ij}(u_{k+1,i} - u_{k,i})(u_{k+1,j} - u_{k,j}), \\ \overline{M}_{ij}(u_{k+1,i} - u_{k,i})(u_{k+1,j} - u_{k,j}) \end{array} \right) \leq 0. \tag{13}$$

The roles of the different conditions may be summarized as follows:

- Conditions (7)-(9) guarantee that all of the constraints be satisfied at $\mathbf{u}_{k+1}$.
- Conditions (10) and (11) guarantee that the scheme not converge prematurely.
- Conditions (12) and (13) guarantee that the cost value always improve from one iteration to the next.

We refer the user to [8, 9] for the justifications of these statements.

3.2 Implementation: Project and Filter

In general, the $\mathbf{u}_{k+1}$ that results from whatever $\mathbf{u}^*_{k+1}$ is provided by a given RTO algorithm and the filter law (3) will not satisfy all of the conditions (7)-(13). As such, the essential implementation of the SCFO consists in generating a modified (projected) input $\bar{\mathbf{u}}^*_{k+1}$ that is, by construction, forced to satisfy Conditions (9)-(12) while remaining as close to $\mathbf{u}^*_{k+1}$ as possible:

$$\begin{array}{rll} \bar{\mathbf{u}}^*_{k+1} = \arg\underset{\mathbf{u}}{\text{minimize}} & \|\mathbf{u} - \mathbf{u}^*_{k+1}\|_2^2 & \\ \text{subject to} & \nabla g_{p,j}(\mathbf{u}_k)^T(\mathbf{u} - \mathbf{u}_k) \leq -\delta_{g,j}, \;\; \forall j : g_{p,j}(\mathbf{u}_k) \geq -\epsilon_{g,j} & \\ & \nabla g_j(\mathbf{u}_k)^T(\mathbf{u} - \mathbf{u}_k) \leq -\delta_{\bar{g},j}, \;\; \forall j : g_j(\mathbf{u}_k) \geq -\epsilon_{\bar{g},j} & , \\ & \nabla \phi_p(\mathbf{u}_k)^T(\mathbf{u} - \mathbf{u}_k) \leq -\delta_\phi & \\ & \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U & \end{array} \tag{14}$$

where $\boldsymbol{\delta}_g, \boldsymbol{\delta}_{\bar{g}}, \delta_\phi, \boldsymbol{\epsilon}_g, \boldsymbol{\epsilon}_{\bar{g}} \succ \mathbf{0}$ are all *projection parameters* used to approximate the conditions. It is easy to show that $\bar{\mathbf{u}}^*_{k+1}$ satisfying these conditions implies that $\mathbf{u}_{k+1}$ satisfies them as well whenever the filter law:

$$\mathbf{u}_{k+1} := \mathbf{u}_k + K_k\left(\bar{\mathbf{u}}^*_{k+1} - \mathbf{u}_k\right) \tag{15}$$

with $K_k \in (0, 1]$ is used.

To satisfy Conditions (7), (8), and (13), one may simply substitute (15) into these conditions and choose K_k such that the resulting expressions hold:

$$g_{p,j}(\mathbf{u}_k) + K_k \sum_{i=1}^{n_u} \max\left(\underline{\kappa}^p_{ji}(\bar{u}^*_{k+1,i} - u_{k,i}), \overline{\kappa}^p_{ji}(\bar{u}^*_{k+1,i} - u_{k,i})\right) \leq 0, \;\; \forall j = 1, ..., n_g \tag{16}$$

$$g_j(\mathbf{u}_k + K_k\left(\bar{\mathbf{u}}^*_{k+1} - \mathbf{u}_k\right)) \leq 0, \;\; \forall j = 1, ..., \bar{n}_g \tag{17}$$

$$K_k \nabla \phi_p(\mathbf{u}_k)^T(\bar{\mathbf{u}}^*_{k+1} - \mathbf{u}_k) + \frac{K_k^2}{2} \sum_{i=1}^{n_u}\sum_{j=1}^{n_u} \max\left(\begin{array}{l} \underline{M}_{ij}(\bar{u}^*_{k+1,i} - u_{k,i})(\bar{u}^*_{k+1,j} - u_{k,j}), \\ \overline{M}_{ij}(\bar{u}^*_{k+1,i} - u_{k,i})(\bar{u}^*_{k+1,j} - u_{k,j}) \end{array}\right) \leq 0, \tag{18}$$

which is guaranteed to be the case for a small enough choice of $K_k > 0$ [8], the latter easily calculated by a line search (see Section 3.8).

It may be proven [8] that applying these conditions leads one to approach a Karush-Kuhn-Tucker (KKT) point to arbitrary closeness as $\boldsymbol{\delta}_g, \boldsymbol{\delta}_{\bar{g}}, \delta_\phi, \boldsymbol{\epsilon}_g, \boldsymbol{\epsilon}_{\bar{g}} \to \mathbf{0}$. The actual choice of these parameters is, however, automated in the solver. Since larger values tend to lead to faster progress but smaller values are needed to force convergence to a KKT point, the solver starts by setting $\boldsymbol{\epsilon}_g := \boldsymbol{\delta}_g := -\underline{\mathbf{G}}_p$, $\boldsymbol{\epsilon}_{\bar{g}} := \boldsymbol{\delta}_{\bar{g}} := -\underline{\mathbf{G}}$, and $\delta_\phi = \max\limits_{i=0,...,k} \hat{\phi}(\mathbf{u}_i) - \underline{\phi}$, and then attempts to solve the projection problem (14) with these parameters. If the problem is found to be infeasible, the projection parameters are halved and the solution is attempted again – this cycle continues until a feasible projection is found. If the parameters have been halved more than a dozen times and still no feasible solution exists, then this is proof that the current input point is a KKT point with very high accuracy. No adaptation of the input is done if this is the case.

3.3 Robust Implementation

Both the projection of (14) and the implicit filter gain limits (16) and (18) are conceptual in nature, as neither the gradients $\nabla g_{p,j}(\mathbf{u}_k), \nabla \phi_p(\mathbf{u}_k)$ nor the exact constraint values $g_{p,j}(\mathbf{u}_k)$ will be available in application – the former being estimated from discrete measurements and the latter either measured with noise or estimated with a certain variance. As such, what is actually used in the SCFO solver is a robust version of these conditions, with the gradients limited to the uncertainty boxes:

$$\begin{array}{l} \nabla \underline{g}_{p,j}(\mathbf{u}_k) \preceq \nabla g_{p,j}(\mathbf{u}_k) \preceq \nabla \overline{g}_{p,j}(\mathbf{u}_k), \quad j = 1, ..., n_g \\ \nabla \underline{\phi}_p(\mathbf{u}_k) \preceq \nabla \phi_p(\mathbf{u}_k) \preceq \nabla \overline{\phi}_p(\mathbf{u}_k) \end{array},$$

and the constraint values upper bounded by the values $\overline{g}_{p,j}(\mathbf{u}_k)$ so that:

$$g_{p,j}(\mathbf{u}_k) \leq \overline{g}_{p,j}(\mathbf{u}_k), \quad j = 1, ..., n_g$$

with sufficiently high confidence.

The robust implementation then becomes:

$$\begin{array}{rl} \bar{\mathbf{u}}^*_{k+1} = \arg\underset{\mathbf{u}}{\text{minimize}} & \|\mathbf{u} - \mathbf{u}^*_{k+1}\|_2^2 \\ \text{subject to} & \nabla \tilde{g}_{p,j}(\mathbf{u}_k)^T(\mathbf{u} - \mathbf{u}_k) \leq -\delta_{g,j}, \quad \forall j : \overline{g}_{p,j}(\mathbf{u}_k) \geq -\epsilon_{g,j} \\ & \forall \nabla \tilde{g}_{p,j}(\mathbf{u}_k) : \nabla \underline{g}_{p,j}(\mathbf{u}_k) \preceq \nabla \tilde{g}_{p,j}(\mathbf{u}_k) \preceq \nabla \overline{g}_{p,j}(\mathbf{u}_k) \\ & \nabla g_j(\mathbf{u}_k)^T(\mathbf{u} - \mathbf{u}_k) \leq -\delta_{\bar{g},j}, \quad \forall j : g_j(\mathbf{u}_k) \geq -\epsilon_{\bar{g},j} \\ & \nabla \tilde{\phi}_p(\mathbf{u}_k)^T(\mathbf{u} - \mathbf{u}_k) \leq -\delta_\phi \\ & \forall \nabla \tilde{\phi}_p(\mathbf{u}_k) : \nabla \underline{\phi}_p(\mathbf{u}_k) \preceq \nabla \tilde{\phi}_p(\mathbf{u}_k) \preceq \nabla \overline{\phi}_p(\mathbf{u}_k) \\ & \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \end{array}, \tag{19}$$

followed by the robust filtering step:

$$\overline{g}_{p,j}(\mathbf{u}_k) + K_k \sum_{i=1}^{n_u} \max\left(\underline{\kappa}^p_{ji}(\bar{u}^*_{k+1,i} - u_{k,i}), \overline{\kappa}^p_{ji}(\bar{u}^*_{k+1,i} - u_{k,i})\right) \leq 0, \quad \forall j = 1, ..., n_g \tag{20}$$

$$K_k \nabla \tilde{\phi}_p(\mathbf{u}_k)^T(\bar{\mathbf{u}}^*_{k+1} - \mathbf{u}_k) + \frac{K_k^2}{2} \sum_{i=1}^{n_u}\sum_{j=1}^{n_u} \max \left(\begin{array}{l} \underline{M}_{ij}(\bar{u}^*_{k+1,i} - u_{k,i})(\bar{u}^*_{k+1,j} - u_{k,j}), \\ \overline{M}_{ij}(\bar{u}^*_{k+1,i} - u_{k,i})(\bar{u}^*_{k+1,j} - u_{k,j}) \end{array} \right) \leq 0 \quad . \tag{21}$$
$$\forall \nabla \tilde{\phi}_p(\mathbf{u}_k) : \nabla \underline{\phi}_p(\mathbf{u}_k) \preceq \nabla \tilde{\phi}_p(\mathbf{u}_k) \preceq \nabla \overline{\phi}_p(\mathbf{u}_k)$$

3.3.1 Tractable Reformulation of the Robust Conditions

As Projection (19) is an optimization problem with a semi-infinite number of constraints, it must be placed into a tractable form in order to be solved. This is done by adding auxiliary variables $\mathbf{s}_\phi \in \mathbb{R}^{n_u}$ and $\mathbf{S} \in \mathbb{R}^{n_g \times n_u}$ and writing the projection problem in the following equivalent form [9]:

$$
\begin{aligned}
\bar{\mathbf{u}}^*_{k+1} = \arg\underset{\mathbf{u},\mathbf{S},\mathbf{s}_\phi}{\text{minimize}} \quad & \left\|\mathbf{u} - \mathbf{u}^*_{k+1}\right\|_2^2 \\
\text{subject to} \quad & \sum_{i=1}^{n_u} s_{ji} \leq -\delta_{g,j} \\
& \left.\frac{\partial \underline{g}_{p,j}}{\partial u_i}\right|_{\mathbf{u}_k} (u_i - u_{k,i}) \leq s_{ji} \\
& \left.\frac{\partial \overline{g}_{p,j}}{\partial u_i}\right|_{\mathbf{u}_k} (u_i - u_{k,i}) \leq s_{ji} \\
& \forall j : \overline{g}_{p,j}(\mathbf{u}_k) \geq -\epsilon_{g,j} \\
& \nabla g_j(\mathbf{u}_k)^T(\mathbf{u} - \mathbf{u}_k) \leq -\delta_{\bar{g},j}, \;\; \forall j : g_j(\mathbf{u}_k) \geq -\epsilon_{\bar{g},j} \\
& \sum_{i=1}^{n_u} s_{\phi,i} \leq -\delta_\phi \\
& \left.\frac{\partial \underline{\phi}_p}{\partial u_i}\right|_{\mathbf{u}_k} (u_i - u_{k,i}) \leq s_{\phi,i} \\
& \left.\frac{\partial \overline{\phi}_p}{\partial u_i}\right|_{\mathbf{u}_k} (u_i - u_{k,i}) \leq s_{\phi,i} \\
& \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U
\end{aligned}
\tag{22}
$$

which is a convex problem with a quadratic objective and linear constraints.

A similar transformation is possible for (21):

$$
\begin{aligned}
K_k \sum_{i=1}^{n_u} \max\left(\left.\frac{\partial \underline{\phi}_p}{\partial u_i}\right|_{\mathbf{u}_k} (\bar{u}^*_{k+1,i} - u_{k,i}), \left.\frac{\partial \overline{\phi}_p}{\partial u_i}\right|_{\mathbf{u}_k} (\bar{u}^*_{k+1,i} - u_{k,i}) \right) \\
+ \frac{K_k^2}{2} \sum_{i=1}^{n_u}\sum_{j=1}^{n_u} \max\left(\begin{array}{l} \underline{M}_{ij}(\bar{u}^*_{k+1,i} - u_{k,i})(\bar{u}^*_{k+1,j} - u_{k,j}), \\ \overline{M}_{ij}(\bar{u}^*_{k+1,i} - u_{k,i})(\bar{u}^*_{k+1,j} - u_{k,j}) \end{array} \right) \leq 0
\end{aligned}
\tag{23}
$$

3.3.2 Choosing the Degree of Robustness in the Gradient Bounds

It will often occur that the robust projection (22) is infeasible due to the bounds on the gradients being too conservative and not tight enough – i.e., one cannot be completely robust and account for all of the possible gradients in the uncertainty sets. When this occurs, the SCFO solver sacrifices some of the robustness so as to make the problem feasible, which it does by artificially making the bounds tighter. Using ϕ_p as an example, and denoting by $\nabla\hat{\phi}_p(\mathbf{u}_k)$ the best gradient estimate at point $\mathbf{u}_k$ (see Section 3.5.3 for how such an estimate may be obtained), the bounds are parameterized as follows:

$$
\begin{aligned}
\nabla\underline{\phi}_p(\mathbf{u}_k) &:= \nabla\hat{\phi}_p(\mathbf{u}_k) + P\Big(\nabla\underline{\phi}_p(\mathbf{u}_k) - \nabla\hat{\phi}_p(\mathbf{u}_k)\Big) \\
\nabla\overline{\phi}_p(\mathbf{u}_k) &:= \nabla\hat{\phi}_p(\mathbf{u}_k) + P\Big(\nabla\overline{\phi}_p(\mathbf{u}_k) - \nabla\hat{\phi}_p(\mathbf{u}_k)\Big)
\end{aligned}
\tag{24}
$$

with $P = 1$ corresponding to full robustness and $P = 0$ corresponding to no robustness, with the latter describing the case where the best gradient estimate is used directly in the projection as if it were the true gradient.

The solver starts with $P = 0$ and attempts to find a choice of projection parameters $\boldsymbol{\delta}_g, \boldsymbol{\delta}_{\bar{g}}, \delta_\phi, \boldsymbol{\epsilon}_g, \boldsymbol{\epsilon}_{\bar{g}}$ so that the projection problem is feasible. In the rare cases that the estimated gradients describe a KKT point this will not be possible and the algorithm will yield $\mathbf{u}_{k+1} = \mathbf{u}_k$ as the solution, with sufficiently low projection parameters being found in most cases. Following this step, the solver then fixes the parameters and begins to increase P as much as possible while retaining the feasibility of the projection. This is done by a bisection algorithm, which then finds $\overline{P}$ as the maximum P value – i.e., the maximum amount of robustness that is possible for the projection with the chosen projection parameters. As this may, in some problems, introduce too much conservatism, a heuristic of choosing $P := 0.5\overline{P}$ is applied, with the resulting P then used to define the bounds in (24), which are in turn used both in the robust projection of (22) and in the robust filter gain limit (23).

3.3.3 Calculating Upper Bounds on Constraint Values

The main way of obtaining robust upper bounds $\overline{g}_{p,j}(\mathbf{u}_k)$ from the measured or estimated values $\hat{g}_{p,j}(\mathbf{u}_k)$ involves supposing an additive noise model:

$$\hat{g}_{p,j}(\mathbf{u}_k) = g_{p,j}(\mathbf{u}_k) + w_{j,k}, \tag{25}$$

and lower bounding the noise to obtain:

$$\overline{g}_{p,j}(\mathbf{u}_k) = \hat{g}_{p,j}(\mathbf{u}_k) - \underline{w}_j, \tag{26}$$

where $\underline{w}_j$ bounds the noise term from below with 99% probability, so that:

$$\text{prob}\left(\underline{w}_j \leq w_{j,k}\right) = 0.99, \tag{27}$$

based on the noise statistics provided by the user (this is calculated in the SCFO solver by Monte Carlo sampling).

A second method involves refining this bound for any repeated measurements, i.e., measurements that correspond to the same point in the input space. In this case, $\mathbf{u}_k = \mathbf{u}_{k+1} = ... = \mathbf{u}_{k-n+1}$, and the bound is refined as:

$$\overline{g}_{p,j}(\mathbf{u}_k) = \frac{1}{n}\sum_{i=k-n+1}^{k} \hat{g}_{p,j}(\mathbf{u}_i) - \underline{w}_j(n), \tag{28}$$

where $\underline{w}_j(n)$ is the lower bound defined as:

$$\text{prob}\left(\underline{w}_j(n) \leq \frac{1}{n}\sum_{i=k-n+1}^{k} w_{j,i}\right) = 0.99. \tag{29}$$

A third bound is calculated using the Lipschitz constants and is derived iteratively from the other upper bounds, (26) and (28), as follows:

$$\overline{g}_{p,j}(\mathbf{u}_k) = \overline{g}_{p,j}(\mathbf{u}) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}_{ji}^{p}(u_{k,i} - u_i), \overline{\kappa}_{ji}^{p}(u_{k,i} - u_i)\right), \tag{30}$$

where $\mathbf{u}$ may be any of the other available data points. As refining one upper bound using this method may, in turn, refine another, the refinement of (30) is looped in the solver, with the loop exiting once no more refinement is obtained (or once the refinement is negligible). Since (26), (28), and (30) all represent valid ways to upper bound the true constraint value, the upper bound that is employed is defined as their minimum.

We note, in passing, that identical methods may be used for lower bounding, with

$$\underline{g}_{p,j}(\mathbf{u}_k) = \hat{g}_{p,j}(\mathbf{u}_k) - \overline{w}_j, \;\; \text{prob}\left(\overline{w}_j \geq w_{j,k}\right) = 0.99, \tag{31}$$

$$\underline{g}_{p,j}(\mathbf{u}_k) = \frac{1}{n}\sum_{i=k-n+1}^{k} \hat{g}_{p,j}(\mathbf{u}_i) - \overline{w}_j(n), \;\; \text{prob}\left(\overline{w}_j(n) \geq \frac{1}{n}\sum_{i=k-n+1}^{k} w_{j,i}\right) = 0.99, \tag{32}$$

$$\underline{g}_{p,j}(\mathbf{u}_k) = \underline{g}_{p,j}(\mathbf{u}) + \sum_{i=1}^{n_u} \min\left(\underline{\kappa}_{ji}^{p}(u_{k,i} - u_i), \overline{\kappa}_{ji}^{p}(u_{k,i} - u_i)\right) \tag{33}$$

serving as the analogues to (26)-(30).

3.4 Management of Noise Statistics

A number of preliminary operations are carried out on the noise statistics that are provided by the user so as to make them compatible with the subroutines where they are required.

3.4.1 Log-Concave Approximation of the Noise PDFs

The current version of the solver requires that the user input all noise statistics via a set of noise samples, as this saves the user the trouble of having to specify a particular model that the solver is compatible with and lets them input the samples directly (perhaps from a set of experimental measurements). However, since a number of the subroutines that use the noise are essentially optimization problems that require the noise probability distribution function (PDF) to be log-concave in order to have tractable, convex problems, a current limitation of the SCFO solver is that it can only work with log-concave PDFs. As such, one of the first steps in the solver is to take the user inputted noise statistics and to approximate them with discretely-defined log-concave functions. This is done via the following steps:

1. Arranging the noise data in histogram form with 100 bins.
2. Adding artificial finite support to the data by populating the left and right sides of the histogram with very small values (10^{-6}).
3. Replacing the histogram values, y_h, with their negative logarithms: $\tilde{y}_h = -\log y_h$.
4. Fitting a convex function to the transformed values $\tilde{y}_h$ to obtain the convex approximation $\tilde{y}_h^{cvx} \approx \tilde{y}_h$.

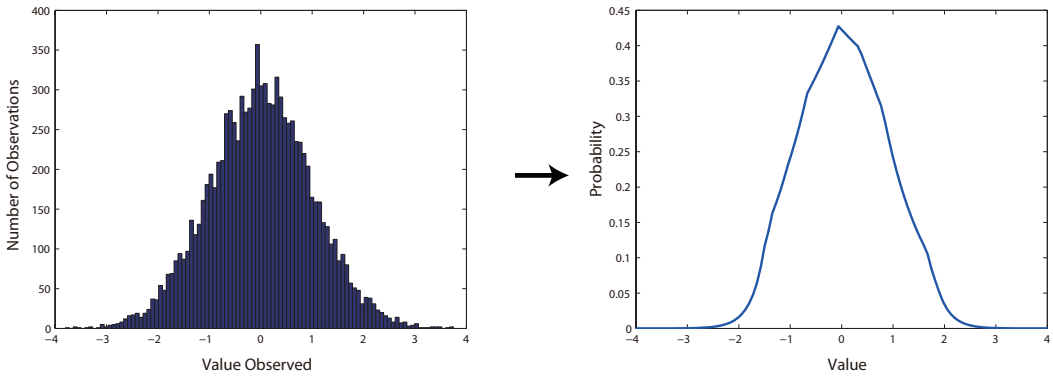

Figure 4: Obtaining a log-concave PDF approximation from noise samples.

5. Taking the exponential of $-\tilde{y}_h^{cvx}$ to obtain the discrete log-concave approximation: $y_h \approx y_h^{lccv} = e^{-\tilde{y}_h^{cvx}}$.
6. Normalizing the approximation (dividing by its integral) to obtain the log-concave PDF approximation.

An illustration of this step is given in Figure 4.

3.4.2 Generating Log-Likelihood Penalty Functions

The gradient estimation subroutines in the SCFO solver (see Section 3.5) employ a maximum-likelihood regularization, for which a log-likelihood penalty function is required. The discrete form of this is simply obtained by taking the negative of the logarithm of the approximate PDF. Because the PDF is forced to be log-concave, the resulting penalty function is always convex (Figure 5). So as to make this function more suitable for numerical solvers (i.e., to allow the problems of interest to be solved as linear programming problems), we take a 5-piece linear approximation of the penalty function, $p_r(x)$:

$$p_r(x) \approx \max\left(a_{r,1}x + b_{r,1}, ..., a_{r,5}x + b_{r,5}\right),$$

which allows us to exploit the following reformulation:

$$\underset{x}{\text{minimize}} \quad p_r(x) \quad \approx \quad \begin{array}{ll} \underset{x,v_1,...,v_5}{\text{minimize}} & \sum_{i=1}^{5} v_i \\ \text{subject to} & a_{r,i}x + b_{r,i} \leq v_i, \quad i = 1, ..., 5 \end{array}, \tag{34}$$

with v denoting slack variables.

3.4.3 Data Binning

The procedure described above is valid for generating the statistical properties of each independent measurement, i.e., the PDF of the noise for each measurement and the corresponding

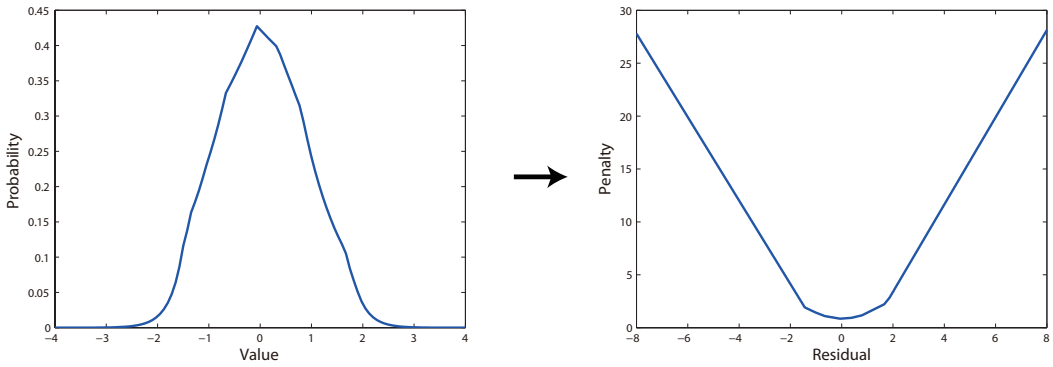

Figure 5: The PDF and its corresponding penalty function.

penalty function. However, there are times when it is more computationally efficient to bin several measurements together – particularly, when measurements have nearly identical $\mathbf{u}$ values, as this allows us to essentially achieve the same results but with a more compact data set whose uncertainty due to noise is reduced. This has also been observed to reduce the number of numerical difficulties encountered by the optimization subroutines.

The basic idea of binning is illustrated geometrically for a simple two-input example in Figure 6. We first present the case where no error is incurred by binning two points together, e.g., when $\mathbf{u}_1 = \mathbf{u}_2$ exactly. Using the cost function as an example, we note that we can easily derive the noise statistics of the averaged point, $\mathbf{u}_{bin}$, via:

$$\begin{aligned}
&\hat{\phi}_p(\mathbf{u}_1) = \phi_p(\mathbf{u}_{bin}) + w_{\phi,1} \\
&\hat{\phi}_p(\mathbf{u}_2) = \phi_p(\mathbf{u}_{bin}) + w_{\phi,2} \\
&\Rightarrow \frac{1}{2}\left(\hat{\phi}_p(\mathbf{u}_1) + \hat{\phi}_p(\mathbf{u}_2)\right) = \phi_p(\mathbf{u}_{bin}) + \frac{1}{2}\left(w_{\phi,1} + w_{\phi,2}\right) \\
&\Leftrightarrow \phi_p(\mathbf{u}_{bin}) = \frac{1}{2}\left(\hat{\phi}_p(\mathbf{u}_1) + \hat{\phi}_p(\mathbf{u}_2)\right) - \frac{1}{2}\left(w_{\phi,1} + w_{\phi,2}\right)
\end{aligned},$$

or, more generally, when n measurements are binned:

$$\phi_p(\mathbf{u}_{bin}) = \frac{1}{n}\sum_{i=1}^{n}\hat{\phi}_p(\mathbf{u}_i) - \frac{1}{n}\sum_{i=1}^{n} w_{\phi,i}. \tag{35}$$

The resulting PDF for $\phi_p(\mathbf{u}_{bin})$, which depends on the mean of the measurements and on the statistics of the stochastic summation term, may be easily approximated by a log-concave distribution following Monte Carlo sampling.

When the points are close but not equal, as in Figure 6, more care should be taken since the act of binning introduces a deterministic error. In this case, we may use the Lipschitz constants to bound the value of the binned measurement as follows:

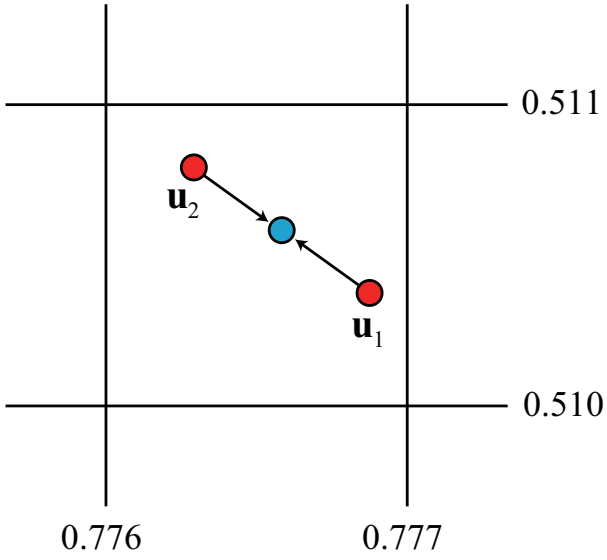

Figure 6: Averaging of two data points (red) that belong to the same bin (the square) to yield a single combined data point (blue).

$$\begin{aligned}
&\phi_p(\mathbf{u}_1) + \sum_{i=1}^{n_u} \min\left(\underline{\kappa}_{\phi,i}(u_{bin,i} - u_{1,i}), \overline{\kappa}_{\phi,i}(u_{bin,i} - u_{1,i})\right) \\
&\quad \leq \phi_p(\mathbf{u}_{bin}) \leq \phi_p(\mathbf{u}_1) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}_{\phi,i}(u_{bin,i} - u_{1,i}), \overline{\kappa}_{\phi,i}(u_{bin,i} - u_{1,i})\right) \\
&\phi_p(\mathbf{u}_2) + \sum_{i=1}^{n_u} \min\left(\underline{\kappa}_{\phi,i}(u_{bin,i} - u_{2,i}), \overline{\kappa}_{\phi,i}(u_{bin,i} - u_{2,i})\right) \\
&\quad \leq \phi_p(\mathbf{u}_{bin}) \leq \phi_p(\mathbf{u}_2) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}_{\phi,i}(u_{bin,i} - u_{2,i}), \overline{\kappa}_{\phi,i}(u_{bin,i} - u_{2,i})\right)
\end{aligned} .$$

We may next substitute in the measurement equality $\phi_p(\mathbf{u}_i) = \hat{\phi}_p(\mathbf{u}_i) - w_{\phi,i}$ to obtain:

$$\begin{aligned}
&\hat{\phi}_p(\mathbf{u}_1) - w_{\phi,1} + \sum_{i=1}^{n_u} \min\left(\underline{\kappa}_{\phi,i}(u_{bin,i} - u_{1,i}), \overline{\kappa}_{\phi,i}(u_{bin,i} - u_{1,i})\right) \\
&\quad \leq \phi_p(\mathbf{u}_{bin}) \leq \hat{\phi}_p(\mathbf{u}_1) - w_{\phi,1} + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}_{\phi,i}(u_{bin,i} - u_{1,i}), \overline{\kappa}_{\phi,i}(u_{bin,i} - u_{1,i})\right) \\
&\hat{\phi}_p(\mathbf{u}_2) - w_{\phi,2} + \sum_{i=1}^{n_u} \min\left(\underline{\kappa}_{\phi,i}(u_{bin,i} - u_{2,i}), \overline{\kappa}_{\phi,i}(u_{bin,i} - u_{2,i})\right) \\
&\quad \leq \phi_p(\mathbf{u}_{bin}) \leq \hat{\phi}_p(\mathbf{u}_2) - w_{\phi,2} + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}_{\phi,i}(u_{bin,i} - u_{2,i}), \overline{\kappa}_{\phi,i}(u_{bin,i} - u_{2,i})\right)
\end{aligned} ,$$

and then combine the two inequalities:

$$\begin{aligned}
&\frac{1}{2}\left(\hat{\phi}_p(\mathbf{u}_1)+\hat{\phi}_p(\mathbf{u}_2)\right)-\frac{1}{2}\left(w_{\phi,1}+w_{\phi,2}\right)\\
&+\frac{1}{2}\Big[\sum_{i=1}^{n_u}\min\left(\underline{\kappa}_{\phi,i}(u_{bin,i}-u_{1,i}),\overline{\kappa}_{\phi,i}(u_{bin,i}-u_{1,i})\right)\\
&\quad+\sum_{i=1}^{n_u}\min\left(\underline{\kappa}_{\phi,i}(u_{bin,i}-u_{2,i}),\overline{\kappa}_{\phi,i}(u_{bin,i}-u_{2,i})\right)\Big]\\
&\qquad\leq\phi_p(\mathbf{u}_{bin})\leq\frac{1}{2}\left(\hat{\phi}_p(\mathbf{u}_1)+\hat{\phi}_p(\mathbf{u}_2)\right)-\frac{1}{2}\left(w_{\phi,1}+w_{\phi,2}\right)\\
&\qquad\quad+\frac{1}{2}\Big[\sum_{i=1}^{n_u}\max\left(\underline{\kappa}_{\phi,i}(u_{bin,i}-u_{1,i}),\overline{\kappa}_{\phi,i}(u_{bin,i}-u_{1,i})\right)\\
&\qquad\quad+\sum_{i=1}^{n_u}\max\left(\underline{\kappa}_{\phi,i}(u_{bin,i}-u_{2,i}),\overline{\kappa}_{\phi,i}(u_{bin,i}-u_{2,i})\right)\Big]
\end{aligned},$$

or, for the general case of n binned measurements:

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}\hat{\phi}_p(\mathbf{u}_i)-\frac{1}{n}\sum_{i=1}^{n}w_{\phi,i}+\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{n_u}\min\left(\underline{\kappa}_{\phi,i}(u_{bin,i}-u_{j,i}),\overline{\kappa}_{\phi,i}(u_{bin,i}-u_{j,i})\right)\\
&\qquad\qquad\leq\phi_p(\mathbf{u}_{bin})\leq\\
&\frac{1}{n}\sum_{i=1}^{n}\hat{\phi}_p(\mathbf{u}_i)-\frac{1}{n}\sum_{i=1}^{n}w_{\phi,i}+\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{n_u}\max\left(\underline{\kappa}_{\phi,i}(u_{bin,i}-u_{j,i}),\overline{\kappa}_{\phi,i}(u_{bin,i}-u_{j,i})\right)
\end{aligned}.$$

Since this only generates lower and upper stochastic bounds on the value of $\phi_p(\mathbf{u}_{bin})$, we cannot give a single PDF as we did for the case of identical $\mathbf{u}$ values. Instead, we are left with a family of possible distributions, defined by a bounded deterministic component and an additive stochastic element. We conclude that the true PDF is somewhere in the family defined by the PDF of (35) with minimal and maximal shifts of $\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{n_u}\min(\underline{\kappa}_{\phi,i}(u_{bin,i}-u_{j,i}),\overline{\kappa}_{\phi,i}(u_{bin,i}-u_{j,i}))$ and $\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{n_u}\max(\underline{\kappa}_{\phi,i}(u_{bin,i}-u_{j,i}),\overline{\kappa}_{\phi,i}(u_{bin,i}-u_{j,i}))$, respectively. This idea is illustrated in Figure 7.

As binning is done solely for improved computational performance of the gradient estimation subroutines, what is particularly relevant is not so much the PDF of the binned data point but its corresponding penalty function (since this is what is actually used in the subroutines). As the PDF is uncertain, the corresponding negative logarithm is as well. However, it is fairly straightforward to obtain a single convex penalty function that encompasses all of the possible penalty functions (see Figure 8). Mathematically, this penalty function may be defined in a piecewise manner by joining the leftmost penalty function, the rightmost penalty function, and the constant corresponding to their minimum.

Practically, this "relaxed" penalty function is used to robustify the gradient-estimation algorithm, as it allows for the algorithm to use binned data without invalidating any of its theoretical properties. The obvious drawback to doing this is that binning points that are very far apart will lead to very large errors and very loose penalty functions, thereby leading to the gradient-estimation algorithms calculating gradient bounds that are not of much use (see Section 3.5 for a supplement to this discussion). For this reason, fairly small bins are used in the solver so as to keep binning error small, with each input direction split into approximately 200 binning compartments (for a total of approximately 200^{n_u} data binning boxes).

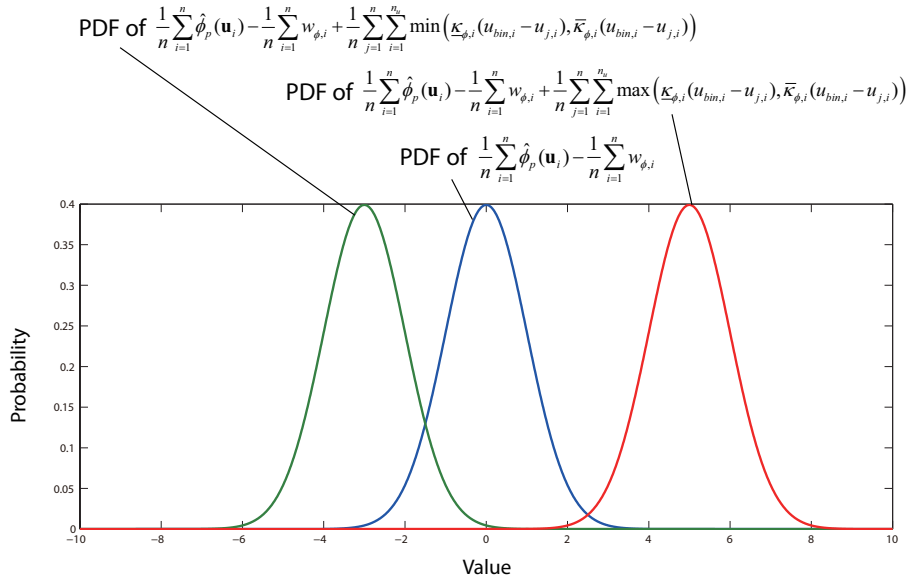

Figure 7: Uncertainty in the PDF of the noise due to binning, with the error-free (no binning error) PDF shown in blue and its corresponding extreme left and right shifts (due to binning error) shown in green and red, respectively. We note that the true PDF must lie somewhere between the green and red extremes. Also, while the two shifts possess different signs here, it is possible for the extreme PDFs to be shifted in the same direction – this depends on the input change due to binning and on the Lipschitz constants.

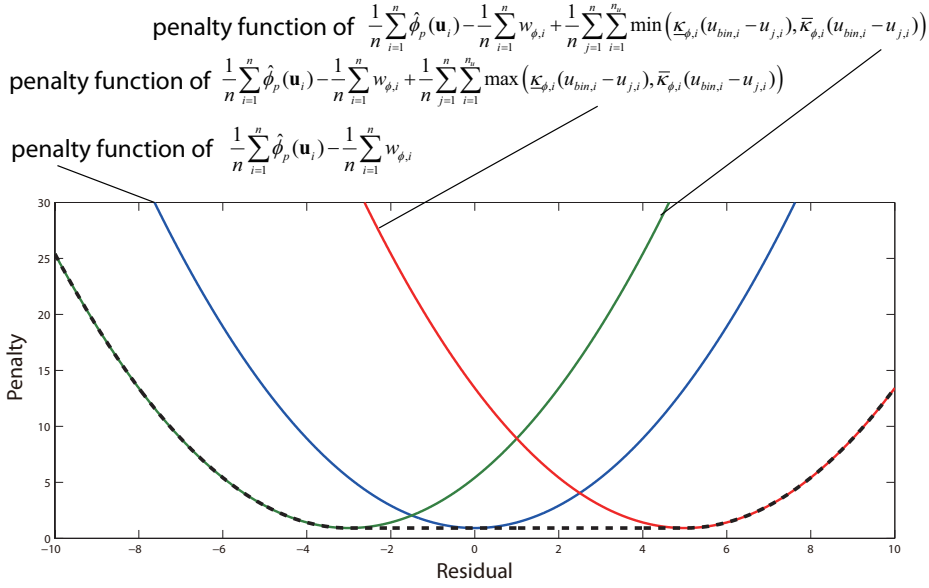

Figure 8: The penalty functions corresponding to the possible PDFs in Figure 7. The dashed black line represents their convex envelope, and is easily seen to underestimate all of the potential penalty functions.

3.5 Gradient Estimation Subroutines

As the SCFO fundamentally depend on the local derivatives of the black-box functions ϕ_p and $\mathbf{G}_p$, a lot of the computational effort in the solver is spent on obtaining "smart" estimates of these derivatives based on the available data and the noise statistics. The essential algorithm at work, as well as the theory behind it, is presented in detail in [5]. As such, we will not go into much detail here and will only outline the crucial aspects, referring the reader to [5] for the full story.

3.5.1 Outline of the Algorithm

The basic purpose of the algorithm is to obtain lower and upper *bounds* on the derivatives for the given reference point, taken here to be $\mathbf{u}_k$. In order to do this, a local structural assumption is made on the function in question, with the current solver automatically deciding between linear, diagonal-quadratic, and full-quadratic structures as follows:

- a linear structure is assumed when the number of measurements provided to the solver is less than $2n_u + 1$,
- a diagonal-quadratic structure (that ignores interaction terms) is assumed when the number of measurements provided to the solver is greater than or equal to $2n_u + 1$ but less than $2n_u + 1 + \sum_{i=1}^{n_u - 1} i$,
- a full-quadratic structure (with interaction terms) is assumed otherwise.

A constrained maximum-likelihood regularization is then carried out in the neighborhood of n points around and including $\mathbf{u}_k$ so as to decide if the chosen structure is a valid fit for the data in question. For the most general (full-quadratic) case, the regularization takes the following form:

$$\begin{array}{ll} \underset{\hat{\mathbf{y}},\hat{\mathbf{w}},\hat{\mathbf{F}},\hat{\mathbf{Q}},\hat{\mathbf{a}},\hat{b}}{\text{minimize}} & \displaystyle\sum_{i=1}^{n} p_{r,i}(\hat{w}_i) \\ \text{subject to} & \hat{w}_i = y_i - \hat{y}_i, \quad i = 1, ..., n \\ & \hat{y}_i = \mathbf{u}_i^T \, \hat{\mathbf{Q}} \, \mathbf{u}_i + \hat{\mathbf{a}}^T \mathbf{u}_i + \hat{b}, \quad i = 1, ..., n \\ & \nabla \hat{f}(\mathbf{u}_i) = 2 \, \hat{\mathbf{Q}} \, \mathbf{u}_i + \hat{\mathbf{a}}, \quad i = 1, ..., n \\ & \underline{\boldsymbol{\kappa}} \preceq \nabla \hat{f}(\mathbf{u}_i) \preceq \overline{\boldsymbol{\kappa}}, \quad i = 1, ..., n \end{array} , \tag{36}$$

with $\hat{\mathbf{y}} \in \mathbb{R}^n$ denoting the estimations of the true outputs, $\mathbf{y} \in \mathbb{R}^n$ the actual measured/estimated outputs, $\hat{\mathbf{w}} \in \mathbb{R}^n$ the estimated additive noise values, $\hat{\mathbf{F}} \in \mathbb{R}^{n_u \times n}$ the estimated gradients, and $\hat{\mathbf{Q}} \in \mathbb{R}^{n_u \times n_u}$, $\hat{\mathbf{a}} \in \mathbb{R}^{n_u}$, $\hat{b}$ the parameters of the quadratic function used to regularize the data (the linear and diagonal-quadratic structures are obtained by simply constraining certain elements of $\hat{\mathbf{Q}}$ to be 0). The penalty functions $p_{r,i}$ are obtained as outlined in Section 3.4, with the transformation of (34) used to make the optimization problem linear. Other regularizing structures are, of course, possible as well [5], and may be included in future versions of the solver.

To judge whether the local structure is statistically invalid or not for a given n measurements, Monte Carlo sampling is employed to calculate the largest acceptable likelihood penalty $\overline{L}$, defined implicitly as:

$$\text{prob}\left(\sum_{i=1}^{n} p_{r,i}(w_i) \leq \overline{L}\right) = 0.99,$$

where the values of w_i are drawn from the corresponding (log-concave) distributions obtained as described in Section 3.4.1.

If the global minimum value of (36) is larger than $\overline{L}$, the structural assumption is rejected as invalid over the neighborhood defined by the n points. Otherwise, it is kept. Using this strategy, a bisection algorithm is employed to find the largest n for which the structure cannot be proven invalid (denoted here by n_L). Once this is done, the minimal and maximal derivative values are calculated subject to this validity constraint by solving two optimization problems for each input (u_i):

$$\begin{array}{ll} \underset{\hat{\mathbf{y}},\hat{\mathbf{w}},\hat{\mathbf{F}},\hat{\mathbf{Q}},\hat{\mathbf{a}},\hat{b}}{\text{minimize}} & \pm\dfrac{\partial \hat{f}}{\partial u_i}\Big|_{\mathbf{u}_k} \\ \text{subject to} & \hat{w}_i = y_i - \hat{y}_i, \quad i = 1, ..., n_L \\ & \hat{y}_i = \mathbf{u}_i^T\, \hat{\mathbf{Q}}\, \mathbf{u}_i + \hat{\mathbf{a}}^T \mathbf{u}_i + \hat{b}, \quad i = 1, ..., n_L \\ & \nabla \hat{f}(\mathbf{u}_i) = 2\, \hat{\mathbf{Q}}\, \mathbf{u}_i + \hat{\mathbf{a}}, \quad i = 1, ..., n_L \\ & \underline{\boldsymbol{\kappa}} \preceq \nabla \hat{f}(\mathbf{u}_i) \preceq \overline{\boldsymbol{\kappa}}, \quad i = 1, ..., n_L \\ & \sum_{i=1}^{n_L} p_{r,i}(\hat{w}_i) \leq \overline{L} \end{array} . \tag{37}$$

The main benefit of this approach is that the calculated bounds are theoretically guaranteed to be accurate (and to contain the true derivatives) for the case when the structural assumption is correct and locally matches the structure of the true function for the n_L points. As we expect many functions to be approximately quadratic in a local neighborhood, this properly is expected to hold approximately in many cases. The obtained bounds may then be used in the robust projection (22).

3.5.2 Extension to the General Input Point

Sometimes, usually due to binning, it is necessary to estimate the gradient for a data point without a corresponding output measurement. To do this, we modify the algorithm of [5] and, denoting the desired reference point by $\mathbf{u}_{ref}$, solve an augmented version of (37):

$$
\begin{aligned}
\underset{\substack{\hat{\mathbf{y}},\, \hat{y}_{ref},\, \hat{\mathbf{w}},\, \hat{\mathbf{F}},\\ \nabla\hat{f}(\mathbf{u}_{ref}),\, \hat{\mathbf{Q}},\, \hat{\mathbf{a}},\, \hat{b}}}{\text{minimize}} \quad & \pm\frac{\partial \hat{f}}{\partial u_i}\Big|_{\mathbf{u}_{ref}} \\
\text{subject to} \quad & \hat{w}_i = y_i - \hat{y}_i, \quad i = 1, ..., n_L \\
& \hat{y}_i = \mathbf{u}_i^T\, \hat{\mathbf{Q}}\, \mathbf{u}_i + \hat{\mathbf{a}}^T \mathbf{u}_i + \hat{b}, \quad i = 1, ..., n_L \\
& \hat{y}_{ref} = \mathbf{u}_{ref}^T\, \hat{\mathbf{Q}}\, \mathbf{u}_{ref} + \hat{\mathbf{a}}^T \mathbf{u}_{ref} + \hat{b} \\
& \nabla\hat{f}(\mathbf{u}_i) = 2\, \hat{\mathbf{Q}}\, \mathbf{u}_i + \hat{\mathbf{a}}, \quad i = 1, ..., n_L \\
& \nabla\hat{f}(\mathbf{u}_{ref}) = 2\, \hat{\mathbf{Q}}\, \mathbf{u}_{ref} + \hat{\mathbf{a}} \\
& \underline{\boldsymbol{\kappa}} \preceq \nabla\hat{f}(\mathbf{u}_i) \preceq \overline{\boldsymbol{\kappa}}, \quad i = 1, ..., n_L \\
& \underline{\boldsymbol{\kappa}} \preceq \nabla\hat{f}(\mathbf{u}_{ref}) \preceq \overline{\boldsymbol{\kappa}} \\
& \sum_{i=1}^{n_L} p_{r,i}(\hat{w}_i) \leq \overline{L}
\end{aligned}
\quad , \tag{38}
$$

which essentially forces the reference point to succumb to the same structural constraints as the others, but does not take its corresponding noise value into consideration for satisfying the bound of $\overline{L}$, as the measurement does not exist.

3.5.3 Obtaining a "Best" Gradient Estimate

As mentioned in Section 3.3.2, a single best estimate of the gradient is needed to act as the center of the gradient uncertainty set. The current version of the solver computes this by simply taking the gradient of the quadratic function with the largest likelihood value for the n_L points:

$$
\begin{aligned}
\underset{\substack{\hat{\mathbf{y}},\, \hat{y}_{ref},\, \hat{\mathbf{w}},\, \hat{\mathbf{F}},\\ \nabla\hat{f}(\mathbf{u}_{ref}),\, \hat{\mathbf{Q}},\, \hat{\mathbf{a}},\, \hat{b}}}{\text{minimize}} \quad & \sum_{i=1}^{n} p_{r,i}(\hat{w}_i) \\
\text{subject to} \quad & \hat{w}_i = y_i - \hat{y}_i, \quad i = 1, ..., n \\
& \hat{y}_i = \mathbf{u}_i^T\, \hat{\mathbf{Q}}\, \mathbf{u}_i + \hat{\mathbf{a}}^T \mathbf{u}_i + \hat{b}, \quad i = 1, ..., n \\
& \hat{y}_{ref} = \mathbf{u}_{ref}^T\, \hat{\mathbf{Q}}\, \mathbf{u}_{ref} + \hat{\mathbf{a}}^T \mathbf{u}_{ref} + \hat{b} \\
& \nabla\hat{f}(\mathbf{u}_i) = 2\, \hat{\mathbf{Q}}\, \mathbf{u}_i + \hat{\mathbf{a}}, \quad i = 1, ..., n \\
& \nabla\hat{f}(\mathbf{u}_{ref}) = 2\, \hat{\mathbf{Q}}\, \mathbf{u}_{ref} + \hat{\mathbf{a}} \\
& \underline{\boldsymbol{\kappa}} \preceq \nabla\hat{f}(\mathbf{u}_i) \preceq \overline{\boldsymbol{\kappa}}, \quad i = 1, ..., n \\
& \underline{\boldsymbol{\kappa}} \preceq \nabla\hat{f}(\mathbf{u}_{ref}) \preceq \overline{\boldsymbol{\kappa}}
\end{aligned}
\quad , \tag{39}
$$

and then outputting the $\nabla\hat{f}(\mathbf{u}_{ref})$ obtained as the best estimate.

3.5.4 Modifications for the Fast Version

If the fast version of the solver is used, what is sacrificed is essentially all of the work done by the gradient estimation subroutines, as these currently take up the majority of the computational effort. In particular, the following simplifications are made:

- Instead of approximating the noise statistics by a general log-concave distribution, they are approximated by a Gaussian distribution with a variance equal to the variance of the provided noise data.
- Instead of solving the regularization problem (36) as a linear programming problem, it is solved via simple least-squares regression (i.e., fitting a quadratic to the data), with the Lipschitz constraints ignored.
- The best estimate is taken as the gradient of the regressed quadratic at the reference point. Any derivatives that violate their corresponding Lipschitz constants are then trimmed to satisfy them.
- The bounds are not calculated and are simply set as the lower and upper Lipschitz constants.
- No data binning is done as this does not affect the speed of the least-squares computations much.

While these simplifications may jeopardize the statistical rigor of the approach (by using a Gaussian distribution) and may not yield very useful bounds on the gradient estimates (by simply using the Lipschitz constants), we note that the observed difference in performance does not appear to be too great in our experience, with the fast version even performing better at times.

3.6 Sufficient-Excitation Considerations

Like many data-driven methods, the SCFO solver requires continued information gathering and is designed to allow perturbations in the inputs with a norm of $\delta_e \in [\underline{\delta}_e, \overline{\delta}_e]$ at all iterations, with the guarantee that any such perturbations do not violate the constraints. This becomes particularly relevant when the solver converges to the neighborhood of the optimum, as the input changes in this region will tend to be very small, or even null, if no excitation is enforced. In order to allow for the asymptotic rejection of the measurement noise and to allow for the optimizer to adapt to potential degradation/changes in the problem over time, it is of interest to keep perturbing the inputs, if only slightly, so as to obtain the most up-to-date local derivative estimates. Additionally, it is desired that the input points generated by the solver be well-poised for regression, as this aids significantly with regard to the quality of these estimates.

Much of the theory related to this topic is taken from [7] (currently in preparation).

3.6.1 Guaranteeing the Existence of a Sufficient-Excitation Ball

It is desired that the ball defined by

$$\mathcal{B}_k = \{\mathbf{u} : \|\mathbf{u} - \mathbf{u}_k\|_2 \leq \underline{\delta}_e\}$$

always lie in the feasible set of the RTO problem, where $\underline{\delta}_e$ is the lower limit on the required excitation, defined in the current version of the solver as:

$$\underline{\delta}_e = \frac{0.005}{n_u} \sum_{i=1}^{n_u} \left(u_i^U - u_i^L\right).$$

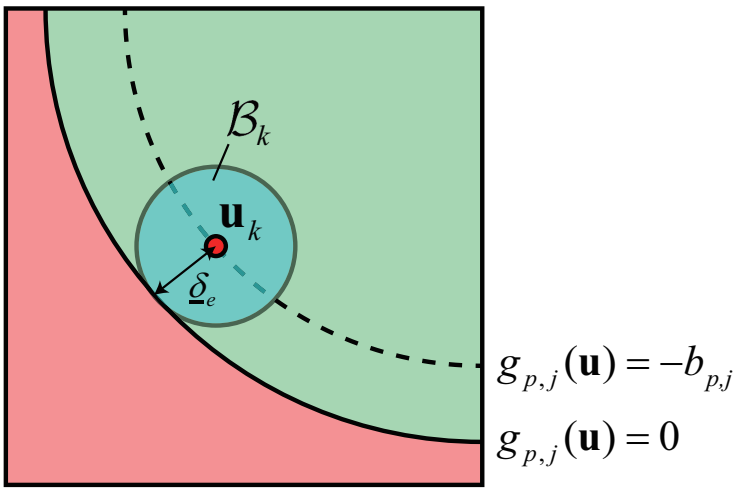

Figure 9: By forcing a back-off on the inequality constraint $g_{p,j}(\mathbf{u}) \leq 0$, it becomes possible to guarantee that the ball $\mathcal{B}_k$ always lie inside the feasible region of the original problem.

In doing this, it is guaranteed that the boundary of $\mathcal{B}_k$, which has the desired sufficient excitation, lie in the feasible set as well. This is accomplished by applying back-offs to the original RTO problem:

$$\begin{array}{ll} \underset{\mathbf{u}}{\text{minimize}} & \phi_p(\mathbf{u}) \\ \text{subject to} & \mathbf{G}_p(\mathbf{u}) \preceq -\mathbf{b}_p \\ & \mathbf{G}(\mathbf{u}) \preceq -\mathbf{b} \\ & \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \end{array} , \tag{40}$$

with the back-offs defined as:

$$\begin{array}{l} b_{p,j} = \underline{\delta}_e \|(\boldsymbol{\kappa}_j^p)^*\|_2, \quad j = 1, ..., n_g \\ b_j = \underline{\delta}_e \|\boldsymbol{\kappa}_j^*\|_2, \quad j = 1, ..., \bar{n}_g \end{array} , \tag{41}$$

where:

$$\begin{array}{l} (\boldsymbol{\kappa}_j^p)^* = \left[\max\left(|\underline{\kappa}_{j1}^p|, |\overline{\kappa}_{j1}^p|\right) \quad ... \quad \max\left(|\underline{\kappa}_{jn_u}^p|, |\overline{\kappa}_{jn_u}^p|\right)\right]^T \\ \boldsymbol{\kappa}_j^* = \left[\max\left(|\underline{\kappa}_{j1}|, |\overline{\kappa}_{j1}|\right) \quad ... \quad \max\left(|\underline{\kappa}_{jn_u}|, |\overline{\kappa}_{jn_u}|\right)\right]^T \end{array} . \tag{42}$$

This is illustrated geometrically in Figure 9 for a simple case with a single constraint. Note that no back-offs are needed for the box constraints due to their orthogonal nature, as even if n_u box constraints are active simultaneously, there will nevertheless exist an orthant that allows for excitation in all directions without violating these constraints (i.e., in the interior of the box).

Conditions (7), (8), (10), and (11) are modified accordingly:

$$g_{p,j}(\mathbf{u}_k) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}_{ji}^p(u_{k+1,i} - u_{k,i}), \overline{\kappa}_{ji}^p(u_{k+1,i} - u_{k,i})\right) \leq -b_{p,j}, \;\; \forall j = 1, ..., n_g \tag{43}$$

$$g_j(\mathbf{u}_{k+1}) \leq -b_j, \;\; \forall j = 1, ..., \bar{n}_g \tag{44}$$

$$\nabla g_{p,j}(\mathbf{u}_k)^T(\mathbf{u}_{k+1} - \mathbf{u}_k) < 0, \;\; \forall j : g_{p,j}(\mathbf{u}_k) + b_{p,j} \approx 0 \tag{45}$$

$$\nabla g_j(\mathbf{u}_k)^T(\mathbf{u}_{k+1} - \mathbf{u}_k) < 0, \;\; \forall j : g_j(\mathbf{u}_k) + b_j \approx 0 \tag{46}$$

with the implemented robust versions re-derived accordingly as explained in Sections 3.2 and 3.3.

3.6.2 Applying the Sufficient-Excitation Condition

In general, one cannot both guarantee satisfaction of the SCFO and sufficient excitation at all iterations, and it may occur that the two are mutually exclusive. Additionally, neither one may be given constant priority, since it may occur that the sufficient-excitation requirements can rarely be met while the SCFO are applied, and applying the SCFO without allowing perturbations to enforce sufficient excitation could lead to poor estimations of the local derivatives.

In the current version of the solver, sufficient excitation is enforced in one of two scenarios:

1. If $\|\mathbf{u}_{k+1} - \mathbf{u}_k\|_2 < \delta_e$ for five or more consecutive iterations, or if $\|\mathbf{u}_{k+1} - \mathbf{u}_k\|_2 \leq 10^{-4}$.
2. If the poisedness metric $\Lambda_k > 10$ for five or more consecutive iterations, with Λ_k defined as the condition number of the matrix:

$$\Delta\mathbf{U}_k := \begin{bmatrix} \tilde{\mathbf{u}}_{k+1-n_u} - \tilde{\mathbf{u}}_{k+2-n_u} \\ \vdots \\ \tilde{\mathbf{u}}_{k+1} - \tilde{\mathbf{u}}_k \end{bmatrix}$$

of scaled inputs:

$$\tilde{u}_{k+1-i,j} := \frac{u_{k+1-i,j} - \min\limits_{i=0,...,n_u} u_{k+1-i,j}}{\max\limits_{i=0,...,n_u} u_{k+1-i,j} - \min\limits_{i=0,...,n_u} u_{k+1-i,j}}, \; i = 0, ..., n_u, \; j = 1, ..., n_u.$$

The first condition is meant to force sufficiently large steps being consistently taken "from time to time" (this frequency decided by the heuristic choice of "five"), while the second is intended to ensure that sufficient excitation is forced whenever there is a consistent linear dependence between consecutive input points. The condition $\|\mathbf{u}_{k+1} - \mathbf{u}_k\|_2 > 10^{-4}$ is enforced to avoid having two consecutive input points being approximately the same, as this provides absolutely no excitation.

The actual choice of δ_e is *adaptive*, since more excitation is desired in regions of the input space with a low signal-to-noise ratio, while less excitation is preferred if the signal-to-noise ratio is

high. In calculating an appropriate value for δ_e, we first fix its domain by specifying $\underline{\delta}_e$ (defined earlier) and $\overline{\delta}_e$, with the latter defined as:

$$\overline{\delta}_e = \min \Delta \mathbf{u}_{max},$$

with $\underline{\delta}_e < \overline{\delta}_e$ implicit.

The next step is to pick a value of δ_e that falls into this domain and is sufficiently large enough to reject potential measurement noise. To do this, we make an estimate of the signal-to-noise ratio for both the cost (if it is noisy) and the uncertain constraint (if they are noisy) functions at $\mathbf{u}_k$. For the noise component, we simply take $\max(|\underline{w}|, |\overline{w}|)$ as an estimate of the worst-case noise magnitude for the appropriate functions. To estimate the signal, we estimate (shown here for the cost only) the expected change in the function value based on a second-order approximation:

$$\frac{\delta_e}{\sqrt{n_u}} \sum_{i=1}^{n_u} \left| \frac{\partial \hat{\phi}_p}{\partial u_i}\Big|_{\mathbf{u}_k} \right| + \frac{1}{2} \frac{\delta_e^2}{n_u} \sum_{i=1}^{n_u} \left| \frac{\partial^2 \hat{\phi}_p}{\partial u_i^2}\Big|_{\mathbf{u}_k} \right|, \tag{47}$$

i.e., we suppose than an "average" excitation will perturb equally in all directions (by $\delta_e/\sqrt{n_u}$), and that the resulting signal, in the best case a sum of positive components, may be approximated by the first and second (diagonal) derivatives multiplied by the appropriate steps. The gradient estimate used is the "best" estimate of Section 3.5.3, while the diagonal second derivatives are estimated by regressing all of the available data with a diagonal-quadratic model. As a heuristic choice, we take $\delta_e \in [\underline{\delta}_e, \overline{\delta}_e]$ as the smallest value for which (47) is greater than half of the worst-case noise magnitude for all functions considered.

For the first insufficient-excitation scenario, where the magnitude of the adaptation steps is consistently small, we first attempt to preserve as much of the SCFO as possible by extending the SCFO-provided $\mathbf{u}_{k+1}$:

$$\mathbf{u}_{k+1} := \mathbf{u}_k + \frac{\delta_e}{\|\mathbf{u}_{k+1} - \mathbf{u}_k\|_2} (\mathbf{u}_{k+1} - \mathbf{u}_k),$$

and see if this $\mathbf{u}_{k+1}$ satisfies $\Lambda_k \leq 10$ and if it may be proven to be feasible for the RTO problem (using the methods of Section 3.8 when uncertain inequality constraints are present). If not, or if $\mathbf{u}_{k+1} := \mathbf{u}_k$, we choose the next input by approximately solving:

$$\begin{aligned} \mathbf{u}_{k+1} := \arg\,\underset{\mathbf{u}}{\text{maximize}} \quad & \min_{i=0,\ldots,k} \|\mathbf{u} - \mathbf{u}_i\|_2 \\ \text{subject to} \quad & \|\mathbf{u} - \mathbf{u}_k\|_2 = \delta_e \end{aligned}, \tag{48}$$

i.e., the sufficiently-exciting input is chosen as the one that is the furthest away, in the Euclidean sense, from all of the inputs that have already been applied. This choice of criterion promotes the exploration of the input space and often encourages poisedness indirectly, as exciting in orthogonal directions that have not received sufficient excitation will generally help maximize this criterion.

To solve (48) approximately, the following randomized procedure is adopted:

1. Five thousand random vectors of n_u elements are generated.

2. These are normalized to yield step directions, which in turn are used to calculate random points on the surface of the corresponding sufficient-excitation ball, all of which satisfy $\|\mathbf{u} - \mathbf{u}_k\|_2 = \delta_e$.
3. The point that has the highest objective value and can be proven to be feasible for the RTO problem is chosen as $\mathbf{u}_{k+1}$.

The resulting $\mathbf{u}_{k+1}$ is then outputted by the solver instead of the standard $\mathbf{u}_{k+1}$ that would have been given normally, together with $\delta_{exit} = 1$ to indicate that the SCFO have been overridden in favor of sufficient excitation. In the case that none of the generated test points can be rigorously proven to meet the RTO problem constraints, δ_e is halved and the procedure is repeated again. If this is repeated until $\delta_e < \underline{\delta}_e$, rigorously feasible test points are guaranteed to be found due to the existence of the sufficient-excitation ball $\mathcal{B}_k$.

When the steps are sufficiently large but the resulting input set is consistently not well-poised, we choose the next input by solving a problem similar to (48), but in the neighborhood of the SCFO-provided $\mathbf{u}_{k+1}$ and subject to the $\Delta\mathbf{u}_{max}$ limitation:

$$\begin{aligned} \mathbf{u}_{k+1} := \arg\underset{\mathbf{u}}{\text{maximize}} \quad & \min_{i=0,\ldots,k} \|\mathbf{u} - \mathbf{u}_i\|_2 \\ \text{subject to} \quad & \|\mathbf{u} - \mathbf{u}_{k+1}\|_2 = \delta_e \\ & \mathbf{u}_k - \Delta\mathbf{u}_{max} \preceq \mathbf{u} \preceq \mathbf{u}_k + \Delta\mathbf{u}_{max} \end{aligned} . \tag{49}$$

3.7 Temporary Constraint Violations

It is often the case that some of the uncertain inequality constraints $\mathbf{G}_p$ in the RTO problem are "soft", in the sense that they should not be violated indefinitely but may be violated temporarily – in particular, while the algorithm is converging to the neighborhood of the optimum. The SCFO solver accomodates such constraints by allowing the user to specify the largest violation that may be allowed for a given constraint, d_j, and the total sum of violations (the "violation integral") that is allowed during all operation, $d_{T,j}$. This allows us to relax the modified Condition (43):

$$g_{p,j}(\mathbf{u}_k) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i}), \overline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i})\right) \leq -b_{p,j} + d_j, \quad \forall j = 1, ..., n_g, \tag{50}$$

which, in many cases, improves convergence speed since it allows for the solver to take bigger steps that would normally be precluded by the need to guarantee feasibility for the 0-slack case.

In order to guarantee that the algorithm does not violate the constraint defined by $d_{T,j}$, the current version of the solver employs the following reduction law:

$$\overline{g}_{p,j}(\mathbf{u}_k) \geq -b_{p,j} \rightarrow d_j := \beta_j d_j, \tag{51}$$

where β_j is set as:

$$\beta_j = \frac{d_{T,j} - d_{j,0}}{d_{T,j}}, \tag{52}$$

with $d_{j,0}$ denoting the original user-set d_j value prior to the reduction of (51). Practically, d_j has a lower tolerance past which it is simply set to 0 (in the current version, this tolerance is 10^{-6}). This guarantees that the constraint can only be violated for a certain (finite) number of iterations before the slack is removed and the original problem, without the slack, is recovered. From a theoretical point of view, this ensures that the desired convergence properties are retained for a sufficiently large number of iterations.

The current version of the solver calculates the appropriate d_j values by cycling through all of the previous measurements and applying (51) as necessary. It is more than likely that this introduces a fair amount of conservatism, as the definition of β_j in (52) is "open-loop" in nature and only employs the initial values. A more advanced reduction law that compares the amount of actual violations incurred to the maximum allowable $d_{T,j}$ would probably improve performance further, and will likely be implemented in future versions of the solver.

3.8 Line Search Techniques and Relaxations

In the solver, the choice of K_k is formulated as a line search problem subject to Conditions (7), (8), (9), (13), and the maximal allowable step changes:

$$
\begin{aligned}
\underset{K_k,\mathbf{u}_{k+1}}{\text{maximize}} \quad & K_k \\
\text{subject to} \quad & \mathbf{u}_{k+1} = \mathbf{u}_k + K_k\left(\bar{\mathbf{u}}^*_{k+1} - \mathbf{u}_k\right) \\
& \overline{g}_{p,j}(\mathbf{u}_k) + \sum_{i=1}^{n_u} \max\left(\begin{array}{l} \underline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i}), \\ \overline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i}) \end{array}\right) \leq -b_{p,j} + d_j,\ j = 1, ..., n_g \\
& g_j(\mathbf{u}_{k+1}) \leq -b_j,\ j = 1, ..., \bar{n}_g \\
& \sum_{i=1}^{n_u} \max\left(\frac{\partial \underline{\phi}_p}{\partial u_i}\Big|_{\mathbf{u}_k}(u_{k+1,i} - u_{k,i}), \frac{\partial \overline{\phi}_p}{\partial u_i}\Big|_{\mathbf{u}_k}(u_{k+1,i} - u_{k,i})\right) \\
& \qquad + \frac{1}{2}\sum_{i=1}^{n_u}\sum_{j=1}^{n_u} \max\left(\begin{array}{l} \underline{M}_{ij}(u_{k+1,i} - u_{k,i})(u_{k+1,j} - u_{k,j}), \\ \overline{M}_{ij}(u_{k+1,i} - u_{k,i})(u_{k+1,j} - u_{k,j}) \end{array}\right) \leq 0 \\
& 0 \leq K_k \leq 1 \\
& \mathbf{u}^L \preceq \mathbf{u}_{k+1} \preceq \mathbf{u}^U \\
& -\Delta\mathbf{u}_{max} \preceq \mathbf{u}_{k+1} - \mathbf{u}_k \preceq \Delta\mathbf{u}_{max}
\end{aligned}
\qquad . \qquad (53)
$$

This problem is solved by a logarithmic line search (with a much finer grid for lower values of K_k and a coarser grid for larger ones).

In this section, we describe several techniques that can relax the constraints in this problem so as to allow for larger values of K_k, which, in general, leads to faster convergence and better performance. Additionally, we discuss the addition of constraints in the line search that prevent a choice of $\mathbf{u}_{k+1}$ that is theoretically guaranteed to *increase* the cost.

3.8.1 Incorporating a Known Cost Function

When the cost function of the RTO problem is known and can be evaluated inexpensively numerically, the objective of (53) is changed and Condition (13) is removed:

$$
\begin{array}{ll}
\underset{K_k, \mathbf{u}_{k+1}}{\text{minimize}} & \phi(\mathbf{u}_{k+1}) \\
\text{subject to} & \mathbf{u}_{k+1} = \mathbf{u}_k + K_k\left(\bar{\mathbf{u}}^*_{k+1} - \mathbf{u}_k\right) \\
& \overline{g}_{p,j}(\mathbf{u}_k) + \sum_{i=1}^{n_u} \max\left(\begin{array}{l} \underline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i}), \\ \overline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i}) \end{array} \right) \leq -b_{p,j} + d_j,\ j = 1, ..., n_g \\
& g_j(\mathbf{u}_{k+1}) \leq -b_j,\ j = 1, ..., \bar{n}_g \\
& 0 \leq K_k \leq 1 \\
& \mathbf{u}^L \preceq \mathbf{u}_{k+1} \preceq \mathbf{u}^U \\
& -\Delta\mathbf{u}_{max} \preceq \mathbf{u}_{k+1} - \mathbf{u}_k \preceq \Delta\mathbf{u}_{max}
\end{array}
. \quad (54)
$$

i.e. the line search is now done with the goal of minimizing the cost instead of maximizing K_k, as the former is linked directly to the degree of optimality obtained, while the latter is more of a heuristic.

3.8.2 Using Concavity Assumptions

When the uncertain inequality constraints may be assumed to be concave in certain variables, the concavity matrix $\mathbf{C}$, as specified by the user, may be used to relax the feasibility guaranteeing constraint as [9]:

$$
\begin{aligned}
& \overline{g}_{p,j}(\mathbf{u}_k) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i}), \overline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i})\right) \leq -b_{p,j} + d_j \\
& \rightarrow \overline{g}_{p,j}(\mathbf{u}_k) + \sum_{i=1}^{n_u} (1 - C_{ji}) \max\left(\underline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i}), \overline{\kappa}^p_{ji}(u_{k+1,i} - u_{k,i})\right) + \\
& \qquad \sum_{i=1}^{n_u} C_{ji} \max\left(\frac{\partial \underline{g}_{p,j}}{\partial u_i}\Big|_{\mathbf{u}_k}(u_{k+1,i} - u_{k,i}), \frac{\partial \overline{g}_{p,j}}{\partial u_i}\Big|_{\mathbf{u}_k}(u_{k+1,i} - u_{k,i}) \right) \leq -b_{p,j} + d_j
\end{aligned}
.
$$

3.8.3 Using All of the Measurements with the Lipschitz Bound

Defining the polyhedron $\mathcal{L}_{k,j}$ as:

$$
\mathcal{L}_{k,j} = \left\{ \mathbf{u} : \overline{g}_{p,j}(\mathbf{u}_k) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}^p_{ji}(u_i - u_{k,i}), \overline{\kappa}^p_{ji}(u_i - u_{k,i})\right) \leq -b_{p,j} + d_j \right\},
$$

we note that any points in this polyhedron are guaranteed to meet the feasibility condition, and that every iteration will generate a valid $\mathcal{L}$. As such, we may define the union of $\mathcal{L}$ over all of the available data points:

$$
\mathcal{L}_{U,j} = \bigcup_{i=0}^{k} \mathcal{L}_{i,j}, \quad (55)
$$

with any $\mathbf{u}_{k+1} \in \mathcal{L}_{U,j}$ satisfying the feasibility condition for $g_{p,j}$. It follows that the intersection of the unions with respect to all of the constraints builds a space that is guaranteed to satisfy the feasibility condition for all of the constraints, thereby allowing us to relax the feasibility condition to:

$$\mathbf{u}_{k+1} \in \bigcap_{j=1}^{n_g} \mathcal{L}_{U,j}.$$

Additionally, we may denote by

$$\mathcal{S}_{\bar{k}} = \left\{ \mathbf{u} : \underline{\phi}_p(\mathbf{u}_{\bar{k}}) + \sum_{i=1}^{n_u} \min\left(\underline{\kappa}_{\phi,i}(u_i - u_{\bar{k},i}), \overline{\kappa}_{\phi,i}(u_i - u_{\bar{k},i})\right) \geq \overline{\phi}_p(\mathbf{u}_k) \right\}$$

a region of the input space where the cost is guaranteed to be superior to the cost at $\mathbf{u}_k$, with the lower and upper bounds $\underline{\phi}_p$ and $\overline{\phi}_p$ being calculated by the methods of Section 3.3.3. Noting that the union of such regions is also undesired, we may prevent the solver from going there by imposing the constraint:

$$\mathbf{u}_{k+1} \notin \bigcup_{\bar{k}=0}^{k-1} \mathcal{S}_{\bar{k}} \,, \tag{56}$$

which is simply added to (53).

3.9 Choice of Reference Point

From a purely theoretical standpoint, the appropriate reference point in the SCFO implementation is always $\mathbf{u}_k$, as it is theoretically guaranteed to be the best point found to date and should be used as the reference so that the SCFO lower the cost further at $\mathbf{u}_{k+1}$ – as choosing any other point as a reference would not guarantee that $\mathbf{u}_{k+1}$ would lead to improvement.

Practically, however, a number of difficulties may arise so as to invalidate the guarantee that $\mathbf{u}_k$ is always the best point to use as a reference. Some of the most frequent are:

- Errors in the gradient estimates, noise in the measurements, or bad Lipschitz bounds lead to an increase in the cost function when applying the SCFO solver, thereby making $\mathbf{u}_k$ a worse point than $\mathbf{u}_{k-1}$ due to $\phi_p(\mathbf{u}_k) > \phi_p(\mathbf{u}_{k-1})$.
- $\mathbf{u}_k$ is a feasible point for the RTO problem with relaxed constraints, but no longer feasible when some of the slack on the constraints has been removed (see Section 3.7).
- $\mathbf{u}_k$ is not a feasible point for the RTO problem with back-offs applied to allow for sufficient excitation (see Section 3.6).
- Previous experimental data, which is independent of the iteration counter k and the RTO algorithm but should nevertheless be provided to the solver, may already possess points better than $\mathbf{u}_k$.
- The problem changes during operation (either due to user-induced or external effects), thereby leading to a problem for which $\mathbf{u}_k$ is no longer the best reference.

In all of these cases, it is of interest to find a reference point other than $\mathbf{u}_k$.

The basic criteria for the reference point, $\mathbf{u}_{ref}$, is that it satisfy all of the constraints robustly (taking into account possible slacks and back-offs – see Sections 3.7 and 3.6) and have the lowest cost. When the cost is uncertain and its measurement/estimation error non-negligible, it may be difficult to decide which point is best, and so preference is given to $\mathbf{u}_k$, then to $\mathbf{u}_{k-1}$, to $\mathbf{u}_{k-2}$, and so on *unless it can be robustly proven that the cost at one of these points is greater than the cost at another*. This may be summarized via the following scheme:

1. Set $\hat{k} := k$.
2. Set $\mathbf{u}_{ref} := \mathbf{u}_{\hat{k}}$.
3. If $\exists i \in \{0, ..., \hat{k}-1\} : \underline{\phi}_p(\mathbf{u}_{ref}) > \overline{\phi}_p(\mathbf{u}_i)$ *or* $\exists j \in \{1, ..., n_g\} : g_{p,j}(\mathbf{u}_{ref}) \geq -b_{p,j} + d_j$ *or* $\exists j \in \{1, ..., \bar{n}_g\} : g_j(\mathbf{u}_{ref}) \geq -b_j$ *or* $\mathbf{u}^L \not\preceq \mathbf{u}_{ref}$ *or* $\mathbf{u}_{ref} \not\preceq \mathbf{u}^U$, set $\hat{k} := \hat{k} - 1$ and return to Step 2. Otherwise, terminate.

This choice of $\mathbf{u}_{ref}$ is then refined further by solving the regularization of (36) in the same manner as done when estimating the gradients – i.e., the solver calculates, by bisection, the largest neighborhood where the quadratic structure is statistically valid. As this regularization smoothes out the noisy data, the final choice of $\mathbf{u}_{ref}$ is the point that the regularization believes to have the lowest cost while robustly satisfying all of the constraints.

Following this choice of $\mathbf{u}_{ref}$, one simply needs to modify Conditions (7)-(13) and their implementation with the substitution $\mathbf{u}_k \to \mathbf{u}_{ref}$.

3.10 Consistency Check for Lipschitz Constants

Taking a constraint $g_{p,j}$ as an example, it follows that the Lipschitz constants satisfying (6) must satisfy the relations:

$$\begin{aligned} g_{p,j}(\mathbf{u}_b) &\leq g_{p,j}(\mathbf{u}_a) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}^p_{ji}(u_{b,i} - u_{a,i}), \overline{\kappa}^p_{ji}(u_{b,i} - u_{a,i})\right) \\ g_{p,j}(\mathbf{u}_b) &\geq g_{p,j}(\mathbf{u}_a) + \sum_{i=1}^{n_u} \min\left(\underline{\kappa}^p_{ji}(u_{b,i} - u_{a,i}), \overline{\kappa}^p_{ji}(u_{b,i} - u_{a,i})\right) \end{aligned} \tag{57}$$

for any two points $\mathbf{u}_a$ and $\mathbf{u}_b$ in the input space. One of the solver's subroutines involves cycling through all of the combinations of available input points (i.e., testing all pairs of $\mathbf{u}_a$ and $\mathbf{u}_b$) and confirming that these relations hold for the Lipschitz constants provided by the user. Since the values of $g_{p,j}$ will usually be corrupted by noise or estimation error, a robust version of (57) is used instead:

$$\begin{aligned} \underline{g}_{p,j}(\mathbf{u}_b) &\leq \overline{g}_{p,j}(\mathbf{u}_a) + \sum_{i=1}^{n_u} \max\left(\underline{\kappa}^p_{ji}(u_{b,i} - u_{a,i}), \overline{\kappa}^p_{ji}(u_{b,i} - u_{a,i})\right) \\ \overline{g}_{p,j}(\mathbf{u}_b) &\geq \underline{g}_{p,j}(\mathbf{u}_a) + \sum_{i=1}^{n_u} \min\left(\underline{\kappa}^p_{ji}(u_{b,i} - u_{a,i}), \overline{\kappa}^p_{ji}(u_{b,i} - u_{a,i})\right) \end{aligned}, \tag{58}$$

as the failure to meet (58) implies the failure to meet (57). In case of such a failure, the Lipschitz constants will be made more conservative by the following algorithm:

1. Set $k_a := 1$.
2. If $k_a \leq 9$, then for $i = 1, ..., n_u$, set $\underline{\kappa}_{ji}^p := 2\underline{\kappa}_{ji}^p$ if $\underline{\kappa}_{ji}^p < 0$ and $\underline{\kappa}_{ji}^p := 0.5\underline{\kappa}_{ji}^p$ if $\underline{\kappa}_{ji}^p > 0$. Likewise, set $\overline{\kappa}_{ji}^p := 2\overline{\kappa}_{ji}^p$ if $\overline{\kappa}_{ji}^p > 0$ and $\overline{\kappa}_{ji}^p := 0.5\overline{\kappa}_{ji}^p$ if $\overline{\kappa}_{ji}^p < 0$. If $k_a > 9$, set $\underline{\kappa}_{ji}^p := -(k_a - 9)^2(\kappa_{ji}^p)^*$ and $\overline{\kappa}_{ji}^p := (k_a - 9)^2(\kappa_{ji}^p)^*$, using the definition in (42).
3. If (58) is not satisfied with the new Lipschitz constants for all available input pairs, set $k_a := k_a + 1$ and return to Step 2. Otherwise, terminate.

This algorithm essentially augments the conservatism of the Lipschitz constants in two stages, with the augmentation being less drastic in the first (for $k_a \leq 9$), as the sign of the user-provided constants is retained while their magnitudes are either increased or decreased by a factor of two. If this fails to yield consistent constants, opposite signs are enforced in the second stage ($k_a > 9$), but the potential magnitude differences with respect to the different input directions are maintained. To ensure that sufficiently conservative constantly are obtained without requiring too much time, the growth in the second stage is squared (by squaring the multiplier).

It has been noted that issues can arise with this routine when $\mathbf{u}_a$ and $\mathbf{u}_b$ are too close together, in that noise statistics that are not entirely robust may lead to differences between two measurements that cannot be compensated purely by the noise and thus lead to the augmentation of the Lipschitz constants. In the case of erroneous noise descriptions, such augmentations can become drastic and lead to arbitrarily large constants the closer that $\mathbf{u}_a$ and $\mathbf{u}_b$ are to one another. To avoid this scenario, which has been noted in practical application, we limit this routine to $\mathbf{u}_a$ and $\mathbf{u}_b$ that do not satisfy the following:

$$-0.1\left(\mathbf{u}^U - \mathbf{u}^L\right) \preceq \mathbf{u}_a - \mathbf{u}_b \preceq 0.1\left(\mathbf{u}^U - \mathbf{u}^L\right).$$

The same procedure is also carried out for the cost ϕ_p, provided that it is uncertain.

4 Application Examples

The purpose of this section is to reinforce the material presented elsewhere in the guide via concrete examples of problems where the SCFO may be applied. Unfortunately, almost all of the examples at the moment are "artificial", in that they deal with simulated RTO problems and not experimental ones, the latter being the problems that the SCFO solver is actually intended for. Some experimental results are available, but are currently in the process of being published and so cannot be presented here (future versions of the guide will, of course, make references to these results). We strongly encourage everyone who has applied the SCFO in the experimental setting to let us know, as we will be more than happy to reference all such work here.

We do note that the simulated examples, though not as satisfying as actual experiments, are nevertheless useful since they represent work that can be easily (in most cases) reproduced by the user. They also allow for analysis of the SCFO performance, since one has knowledge of the solution (the true optimum) in many of these cases and may see how close the SCFO come to achieving it.

In the five application examples that follow, we will first show how the original problem may be stated as an RTO one, how the SCFO solver may be configured to solve it, and, finally, provide the results of the solver's performance. Whenever possible, we also provide brief qualitative descriptions of forthcoming experimental applications. By default, we will use the standard (as opposed to fast) version of the solver, although we note that these examples have been tested with the fast version and often give comparable results.

4.1 Two-Layer Steady-State Optimization

A context where RTO is particularly popular is that of process systems engineering, where the RTO represents a layer in a hierarchy. Often, this hierarchy may be simplified to the two layers of particular technical interest – the optimization layer and the control layer below it [2, 14, 24], as shown in Figure 10. Here, some sort of "plant" is to be operated in a manner that ensures its safety while maximizing profits. Usually, the operation is dynamic in nature, with a number of controller schemes ensuring that specified steady-state setpoints are tracked and met. These setpoints are in turn chosen by the optimizer in an economically optimal way, subject to the stated user/market demand or targets that are in turn specified by a higher management layer. Measurements from the plant are both used for the purposes of feedback control as well as for optimization.

4.1.1 Formulation as an RTO Problem

The common practice in the two-layer scheme is to optimize only at steady state, i.e., to have the optimizer output some setpoints, let the controller achieve those setpoints, and then wait for the system to reach equilibrium, after which steady-state measurements are collected and used to re-optimize the setpoints for the controllers. The benefit of this approach is that the dynamics are largely removed. However, an important assumption on the existence of a steady-state mapping is still required. In particular, one needs the guarantee that applying the same setpoints will always lead to the same steady-state output values (plus or minus measurement noise), which need not be the case in practice but may nevertheless hold in many applications.

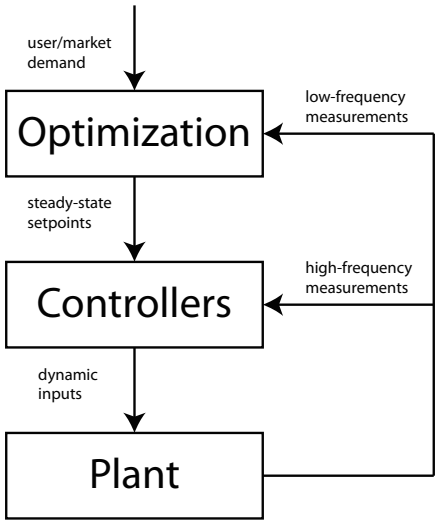

Figure 10: A basic illustration of the two-layer scheme with a dynamic control layer and a steady-state optimization layer.

When this assumption does hold, the formulation of the RTO problem of form (1) is fairly straightforward, as one simply defines $\mathbf{u}$ to be the setpoints for the controllers, ϕ_p as the negative of the plant profit, and $\mathbf{G}_p$ and $\mathbf{G}$ as any safety or user-imposed constraints on the plant. A particular characteristic of such problems is that a model of the uncertain functions ϕ_p and $\mathbf{G}_p$ will usually be available, which, as will be shown in the simulated example, may ease the choice of Lipschitz and quadratic bound constants significantly. Another important characteristic is the existence of a large number of dynamic measurements, which make it easier both to characterize the measurement noise $\mathbf{w}$ and $\mathbf{W}_g$ and to filter out a lot of the error at steady state with the help of estimators.

4.1.2 Simulation Example: Williams-Otto Reactor

A common simulation example for the two-layer steady-state RTO is the Williams-Otto plant [30], which may be described as an ideal continuous-stirred tank reactor (CSTR) with the following reactions taking place:

$$\begin{aligned} A + B &\xrightarrow{k_1} C, & k_1 &= 1.660 \times 10^6 e^{-6666.7/(T_R+273.15)} \\ C + B &\xrightarrow{k_2} P + E, & k_2 &= 7.212 \times 10^8 e^{-8333.3/(T_R+273.15)} \\ C + P &\xrightarrow{k_3} G, & k_3 &= 2.675 \times 10^{12} e^{-11111/(T_R+273.15)} \end{aligned} \quad , \tag{59}$$

where reactants A and B are fed into the reactor with flows F_A and F_B. For this particular example, we assume F_A to be fixed at 1.8275 kg/s, the reactor to have a mass holdup of 2105 kg, and the operation to be isothermal at the temperature T_R.

The goal of the RTO is to vary the setpoints $\mathbf{u} := [F_B \;\; T_R]^T$ so as to optimize the steady-state behavior of the dynamic system that results from (59), with the following steady-state profit function maximized:

$$\begin{aligned}&\text{Profit} := 1143.38X_P(F_A+F_B)+25.92X_E(F_A+F_B)-76.23F_A-114.34F_B\\&\rightarrow \phi_p(\mathbf{u}) = (-1143.38X_P(\mathbf{u})-25.92X_E(\mathbf{u}))(F_A+u_1)+114.34u_1\end{aligned},$$

with X denoting the steady-state concentrations of the products/reactants (implicitly functions of $\mathbf{u}$).

A simpler two-reaction model of the real system is assumed to be available [16]:

$$\begin{aligned}&A+2B \xrightarrow{k_1^*} P+E, \qquad && k_1^* = 2.189\times 10^8 e^{-8077.6/(T_R+273.15)}\\&A+B+P \xrightarrow{k_2^*} G, && k_2^* = 4.310\times 10^{13} e^{-12438/(T_R+273.15)}\end{aligned},$$

which will not, due to plant-model error, yield the plant optimum if optimized but will nevertheless be useful for obtaining basic information about the process.

Choosing to vary F_B between 3 and 6 kg/s and T_R between 70 and 100°C, we have the RTO problem in standard form as:

$$\begin{aligned}\underset{\mathbf{u}}{\text{minimize}} \quad & (-1143.38X_P(\mathbf{u})-25.92X_E(\mathbf{u}))(F_A+u_1)+114.34u_1 && \left.\vphantom{\begin{matrix}a\\b\end{matrix}}\right\}\phi_p(\mathbf{u})\\ \text{subject to} \quad & 3\le u_1\le 6 && \left.\vphantom{\begin{matrix}a\\b\end{matrix}}\right\}\mathbf{u}^L\preceq\mathbf{u}\preceq\mathbf{u}^U\\ & 70\le u_2\le 100\end{aligned},$$

which is then scaled as:

$$\begin{aligned}&\phi_p(\mathbf{u}) := \phi_p(\mathbf{u})/300\\&u_1 := u_1/10\\&u_2 := u_2/100\end{aligned}.$$

The following solver configurations are used.

Initialization

The initial point is taken as $\mathbf{u}_0 := [4.8 \quad 77]^T$ as this corresponds, roughly, to the model-based optimum of the problem (Figure 11). It is generally advised to initialize the SCFO solver with at least n_u+1 data points to have proper functionality (at least n_u+1 data points are needed to obtain an estimate of the local gradient), and so the following "smart initialization" is used to generate the remaining n_u points following the application of $\mathbf{u}_0$:

1. Initialize $\mathbf{S}\in\mathbb{R}^{n_u\times n_u}$ as a diagonal matrix with $S_{11}:=1$ and all other elements set to 0. Set $k:=1$.
2. Define $\mathbf{u}_k := \mathbf{u}_0+\mathbf{S}\Delta\mathbf{u}_{max}$ and calculate the following matrix:

$$\Delta\mathbf{U} := \begin{bmatrix}(\mathbf{u}_0-\mathbf{u}_1)^T\\(\mathbf{u}_1-\mathbf{u}_2)^T\\\vdots\\(\mathbf{u}_{k-1}-\mathbf{u}_k)^T\end{bmatrix}.$$

If the condition number of $\Delta\mathbf{U}$ is above 50, re-define $\mathbf{u}_k$ as $\mathbf{u}_k := \mathbf{u}_{k-1} + \mathbf{S}_k \Delta\mathbf{u}_{max}$, where $\mathbf{S}_k$ is a diagonal matrix of zeros with the sole k^{th} diagonal element equal to 1.

3. Obtain the corresponding $\hat{\phi}_p(\mathbf{u}_k)$ by applying $\mathbf{u}_k$ to the system at hand. Define:

$$\Delta\Phi := \begin{bmatrix} \hat{\phi}_p(\mathbf{u}_0) - \hat{\phi}_p(\mathbf{u}_1) \\ \hat{\phi}_p(\mathbf{u}_1) - \hat{\phi}_p(\mathbf{u}_2) \\ \vdots \\ \hat{\phi}_p(\mathbf{u}_{k-1}) - \hat{\phi}_p(\mathbf{u}_k) \end{bmatrix},$$

and calculate:

$$\nabla\hat{\phi}_p := (\Delta\mathbf{U})^{\dagger}\, \Delta\Phi,$$

with † denoting the Moore-Penrose pseudoinverse.

4. Re-define $\mathbf{S}$ as a diagonal matrix with the diagonals set as:

$$S_{ii} := \begin{cases} 1 & \nabla\hat{\phi}_{p,i} \leq 0 \text{ and } i \leq k \\ -1 & \nabla\hat{\phi}_{p,i} > 0 \text{ and } i \leq k \\ 1 & i = k+1 \\ 0 & i > k+1 \end{cases}.$$

5. Set $k := k + 1$. If $k > n_u$, terminate. Otherwise, return to Step 2.

The motivation behind this initialization is the idea that one can begin to optimize while building the initial data set. This is done by augmenting the perturbations one input at a time so as to isolate the contributions of the different inputs and to guide future perturbations in those directions that seem more promising with respect to the cost. The perturbation steps of $\Delta\mathbf{u}_{max}$ are chosen so as to help reject the noise in the estimate of $\nabla\phi_p$. An additional step is used to control that the perturbation matrix $\Delta\mathbf{U}$ does not become ill-conditioned, which could blow up the estimation error significantly when the problem is of modest size (say, $n_u > 5$) and the measurement error is not negligible.

We note that we choose $\Delta\mathbf{u}_{max} := [0.3 \quad 3]$ ($[0.03 \quad 0.03]$ after scaling) in this example, as these correspond to 10% of the input domain for the two inputs. The costs ϕ_p are obtained by simulating the system of ODEs (the mole balances) that result from (59), as this yields the steady-state concentration values for the given $\mathbf{u}$. A measurement error of $w_\phi \sim \mathcal{N}(0, 0.25)$ is added to obtain the $\hat{\phi}_p$ values.

Nominal RTO Target

The nominal RTO target, $\mathbf{u}^*_{k+1}$, is chosen by implementing a simple gradient-descent step (with a diminishing step size):

$$\mathbf{u}^*_{k+1} := \mathbf{u}_k - \frac{1}{k}\nabla\hat{\phi}_p(\mathbf{u}_k),$$

where the estimate $\nabla\hat{\phi}_p(\mathbf{u}_k)$ is obtained by the method of Section 3.5.3.

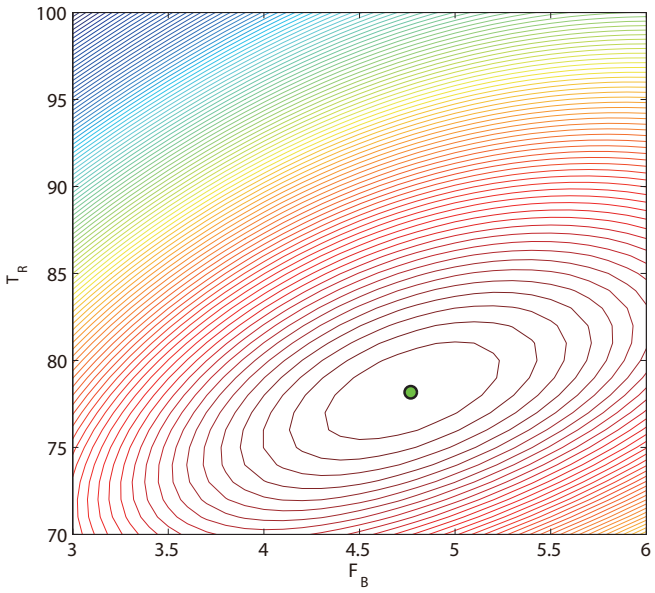

Figure 11: Profit contour plot generated by the model of the Williams-Otto process, with the model optimum shown in green.

Noise Models

The noise model for the (scaled) cost is assumed to be known, and is specified as $\mathcal{N}(0, 0.25/300^2)$.

Lipschitz Constants of the Scaled Cost Function

The Lipschits constants for the scaled cost are estimated as follows:

1. Calculate the Lipschitz constants of the model by gridding, using finite differences to compute the derivatives at each point, and then taking their minimal and maximal values (this is shown in Figure 12, where the minimal and maximal derivatives are -0.3 and 0.2 with respect to u_1 and -0.04 and 0.04 with respect to u_2).
2. Account for input scaling by multiplying the obtained derivatives by the scaling magnitude used. This leads to -3 and 2 with respect to u_1 and -4 and 4 with respect to u_2.
3. Add conservatism to these estimates (to account for plant-model mismatch) by doubling the lower Lipschitz constants if they are negative and cutting them in half if they are positive, and doubling the upper Lipschitz constants if they are positive and cutting them in half if they are negative. This leads to:

$$\underline{\kappa}_{\phi,1} := -6,\ \overline{\kappa}_{\phi,1} := 4,\ \underline{\kappa}_{\phi,2} := -8,\ \overline{\kappa}_{\phi,2} := 8.$$

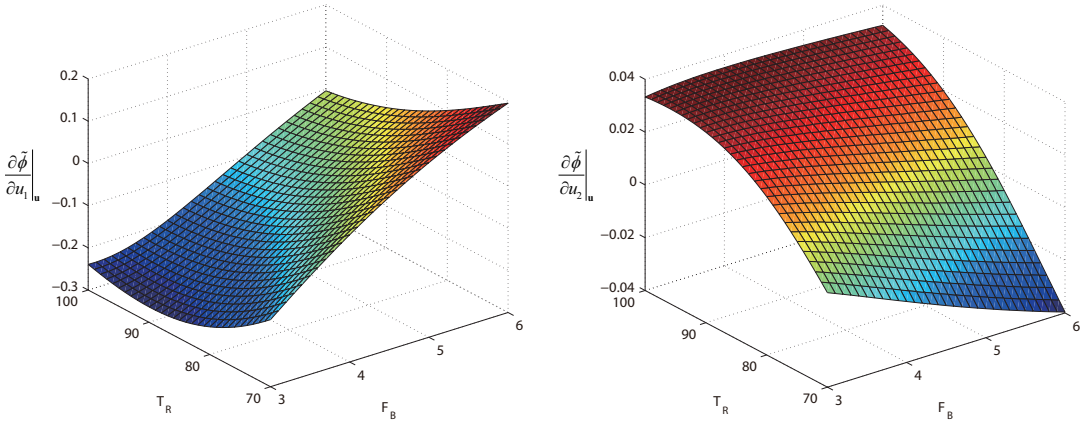

Figure 12: The surface plots for the derivatives of the (scaled) cost model.

Quadratic Bound Constants

The quadratic bound constants are estimated in a similar fashion:

1. Calculate the second-order derivative bounds of the model by gridding, using finite differences to compute the second derivatives at each point, and then taking their minimal and maximal values (this is shown in Figure 13, where the minimal and maximal derivatives are -0.05 and 0.2 with respect to u_1^2, -0.06 and 0.02 with respect to u_1u_2, and -0.001 and 0.004 with respect to u_2^2).
2. Account for input scaling by multiplying the obtained derivatives by the scaling magnitude used *squared* (or by the two magnitudes multiplied for the mixed term). This leads to -5 and 20 with respect to u_1^2, -60 and 20 with respect to u_1u_2, and -10 and 40 with respect to u_2^2.
3. Add conservatism to these estimates as done for the Lipschitz constants. This leads to:

$$\underline{M}_{11} := -10,\ \overline{M}_{11} := 40,\ \underline{M}_{12} = \underline{M}_{21} := -120,\ \overline{M}_{12} = \overline{M}_{21} := 40,\ \underline{M}_{22} := -20,\ \overline{M}_{22} := 80.$$

Global Minimum Cost and Convergence Tolerance

The global minimum cost was set as $\underline{\phi} := -2/3$ (-200 pre-scaling). A tolerance of $\epsilon_\phi := 0$ was used for simplicity (i.e., we preferred to let the algorithm run indefinitely without a stopping criterion).

The problem is solved and the results are reported in Figure 14. We see that the solver reliably brings the operating conditions to a reasonably tight neighborhood of the optimum and keeps them there.

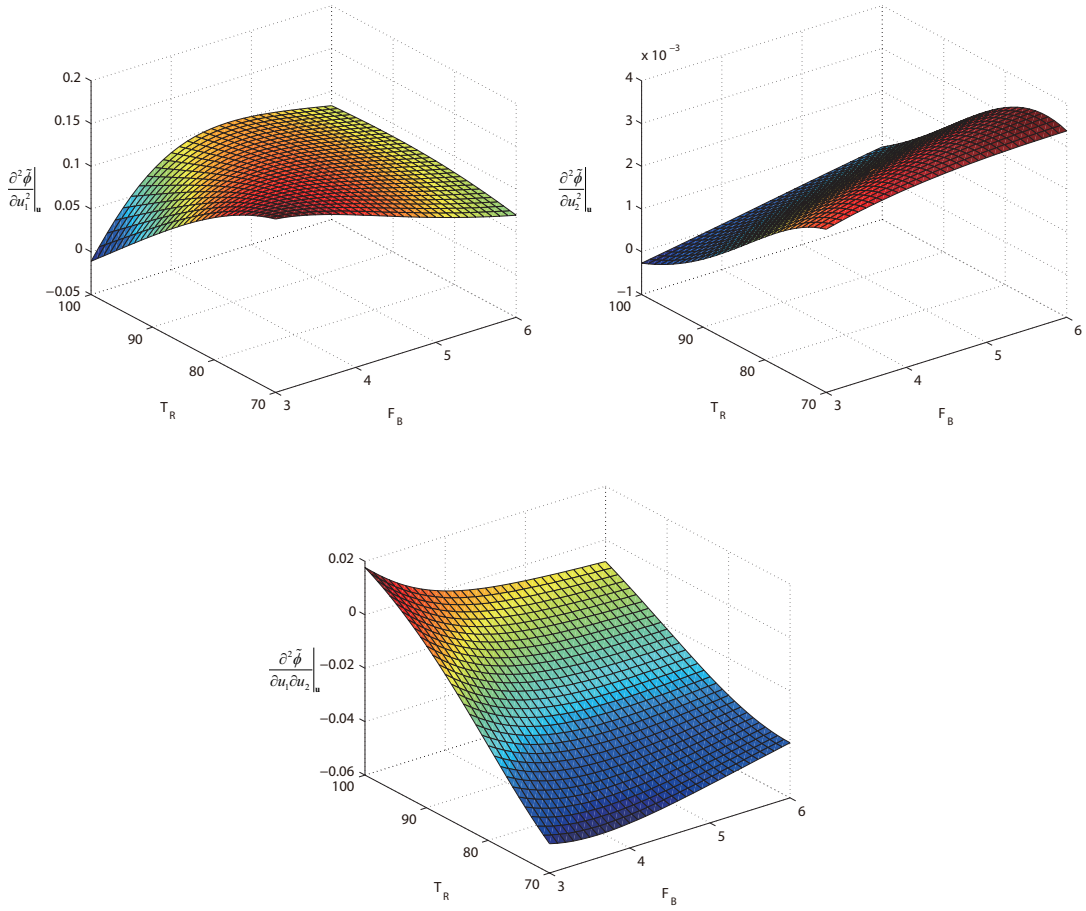

Figure 13: The surface plots for the second derivatives of the (scaled) cost model.

4.1.3 Experimental Example: Fuel Cell Efficiency (Description Only)

Plans to apply the SCFO solver to a solid-oxide fuel cell system are currently underway. Here, the problem involves maximizing the steady-state efficiency of the fuel cell system while meeting a user-specified power demand and satisfying an upper limit on cell potential so as to avoid major degradative effects [11]. Doing this for a simple system where the inputs consists of the fuel and air fluxes, as well as a control on the current of electrons from the anode to the cathode (see Figure 15), leads to the following problem:

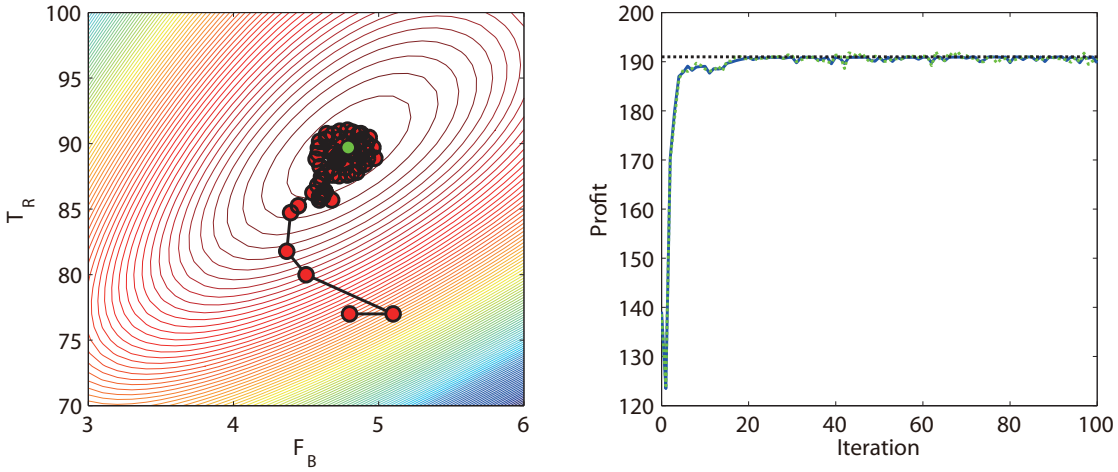

Figure 14: Performance of the SCFO solver for the Williams-Otto simulated problem.

$$
\begin{aligned}
\underset{\dot{n}_{H_2},\dot{n}_{O_2},I}{\text{maximize}} \quad & \eta(\mathbf{u}) \\
\text{subject to} \quad & U_{cell}(\dot{n}_{H_2},\dot{n}_{O_2},I) \geq 0.75\text{V}, \ \forall\text{cells} \\
& p_{el}(\dot{n}_{H_2},\dot{n}_{O_2},I) \geq p_{el}^{S} \\
& \nu(\dot{n}_{H_2},\dot{n}_{O_2},I) \leq 0.75 \\
& 4 \leq 2\frac{\dot{n}_{O_2}}{\dot{n}_{H_2}} \leq 7 \\
& 3.14\text{ml}/(\text{min}\cdot\text{cm}^2) \leq \dot{n}_{H_2} \leq \dot{n}_{H_2}^{U} \\
& 6.28\text{ml}/(\text{min}\cdot\text{cm}^2) \leq \dot{n}_{O_2} \leq 2\dot{n}_{H_2}^{U} \\
& 0\text{A} \leq I \leq 30\text{A}
\end{aligned}
\quad , \tag{60}
$$

with $\dot{n}_{H_2}$ being the hydrogen flux, $\dot{n}_{O_2}$ the oxygen flux, and I the current. η denotes the efficiency of the fuel cell system and is an unknown (though modeled) function of the inputs. U_{cell} is the intercell potential that must be kept above a certain threshold (0.75 volts) so as not to promote cell degradation, while p_{el} is the power produced by the cell, which must be at least the amount specified by the user, p_{el}^{S}. It is noted that both U_{cell} and p_{el} are unknown (though modeled) inequality constraints. ν is the fuel utilization ratio, and must be kept below a certain value to avoid cell starvation (which could easily destroy the cell) – it is an algebraic relation and is known with *certainty*. The oxygen-to-hydrogen ratio is constrained to be between 4 and 7 for this system so as to avoid major thermal gradients, and the hydrogen flux is to be kept above a certain lower limit (so as not to risk starvation), with an upper limit that is not crucial and should be user defined depending on the particular system. The oxygen-to-hydrogen ratio then leads to implicit lower and upper bounds on the oxygen flux, and the current is constraint to be less than 30A so as to avoid overheating the stack, with the lower constraint of 0A implicit.

Defining $\mathbf{u} := [\dot{n}_{H_2} \ \dot{n}_{O_2} \ I]^T$ and rearranging allows us to pose (60) as an RTO problem in standard form:

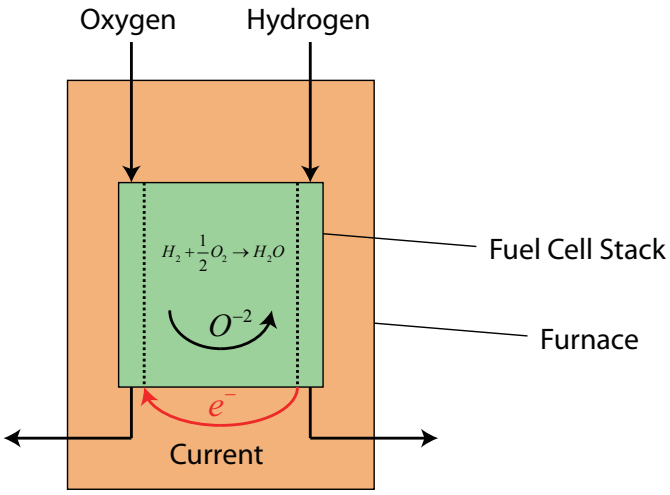

Figure 15: The schematic of a simple solid-oxide fuel cell system.

$$
\begin{array}{lll}
\underset{\mathbf{u}}{\text{minimize}} & -\eta(\mathbf{u}) & \left.\vphantom{\begin{array}{l}a\\a\\a\\a\\a\\a\\a\\a\end{array}}\right\} \phi_p(\mathbf{u}) \\
\text{subject to} & -U_{cell}(\mathbf{u}) + 0.75 \leq 0, \ \forall \text{cells} & \left.\vphantom{\begin{array}{l}a\\a\\a\\a\\a\\a\\a\\a\end{array}}\right\} \mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0} \\
& -p_{el}(\mathbf{u}) + p_{el}^S \leq 0 & \\
& \nu(\mathbf{u}) - 0.75 \leq 0 & \left.\vphantom{\begin{array}{l}a\\a\\a\\a\\a\\a\\a\\a\end{array}}\right\} \mathbf{G}(\mathbf{u}) \preceq \mathbf{0} \\
& 2u_2 - 7u_1 \leq 0 & \\
& 4u_1 - 2u_2 \leq 0 & \\
& 3.14 \leq u_1 \leq \dot{n}_{H_2}^U & \left.\vphantom{\begin{array}{l}a\\a\\a\\a\\a\\a\\a\\a\end{array}}\right\} \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \\
& 6.28 \leq u_2 \leq 2\dot{n}_{H_2}^U & \\
& 0 \leq u_3 \leq 30 &
\end{array}
\quad .
$$

4.2 Run-to-Run Dynamic Optimization of Batch Processes

An interesting RTO problem arises in the finite-parameter reformulation of a dynamic optimization problem, where the goal is to find an optimal dynamic profile over a finite-time batch subject to certain (equally dynamic) constraints. In the most general case, this is usually stated as an optimal control problem having the form:

$$
\begin{array}{ll}
\underset{\mathbf{x}(t),\mathbf{u}(t),t_f}{\text{minimize}} & \Phi\left(\mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f\right) + \displaystyle\int_{t_0}^{t_f} \mathcal{L}\left(\mathbf{x}(t), \mathbf{u}_d(t), t\right) dt \\
\text{subject to} & \dot{\mathbf{x}}(t) = \mathbf{g}_{dyn}\left(\mathbf{x}(t), \mathbf{u}_d(t), t\right) \\
& \mathbf{g}_{path}\left(\mathbf{x}(t), \mathbf{u}_d(t), t\right) \preceq \mathbf{0} \\
& \mathbf{g}_{bound}\left(\mathbf{x}(t_0), t_0, \mathbf{x}(t_f), t_f\right) = \mathbf{0}
\end{array}
\quad , \qquad (61)
$$

where Φ and $\mathcal{L}$ are the endpoint cost and Lagrangian, $\mathbf{x}$ denotes the states of the problem, $\mathbf{u}_d$

the dynamic input, and $\mathbf{g}_{dyn}$, $\mathbf{g}_{path}$, and $\mathbf{g}_{bound}$ the dynamic constraints, the path constraints, and the boundary conditions, respectively.

Such problems are known to be difficult to solve even numerically, and are naturally even harder when the functions involved are unknown, as is generally the case in the optimization of experimental batch processes. However, it is possible to approximate such problems by parameterizing the inputs $\mathbf{u}_d(t)$ in some finite manner and optimizing over the parameters, as the batch will often be run many times and so one may adapt the parameters from batch to batch based on the quality of the product obtained from each batch.

4.2.1 Reformulation as an RTO Problem

There exist multiple ways to parameterize a given $\mathbf{u}_d(t)$ by a finite set of parameters. In this section, we will consider three such approaches. To illustrate them concretely, we will apply the parameterizations to the dynamic problem of minimizing the batch time of a batch polymerization (taken from [17]), where the temperature profile of the jacketed reactor, $T(t)$, is to be optimized together with the batch time, t_f, subject to certain safety and performance constraints. As an optimal control problem, this is stated as:

$$\begin{array}{ll} \underset{T(t),t_f}{\text{minimize}} & t_f \\ \text{subject to} & \text{mass, molar, and energy balances (ODEs)} \\ & T^L \leq T(t) \leq T^U, \ \ \forall t \in [0, t_f] \\ & T_j^L \leq T_j(t), \ \ \forall t \in [0, t_f] \\ & X(t_f) \geq X_d \\ & M_n(t_f) \geq M_{n,d} \end{array} , \tag{62}$$

where T_j denotes the jacket temperature, X denotes the conversion, and M_n the molecular weight, the latter two needing to be above certain desired quantities (subscripted by d) at the end of the batch.

Prior to parameterizing the problem, we make some simplifications. We first note that the dynamic balances and the setup of the problem (where $T_j(t)$ is used to control $T(t)$) imply a relationship between $T(t)$ and $T_j(t)$, which we will denote as $T_j(t) = f_T(T(t))$. Both the concentration X and the molecular weight M_n at the final time t_f also depend, implicitly, on these balances and on both $T(t)$ and t_f, leading us to define the implicit functions $X(t_f) = X_{t_f}(T(t), t_f)$ and $M_n(t_f) = M_{n,t_f}(T(t), t_f)$. Having done this, we may rewrite the problem as:

$$\begin{array}{ll} \underset{T(t),t_f}{\text{minimize}} & t_f \\ \text{subject to} & T^L \leq T(t) \leq T^U, \ \ \forall t \in [0, t_f] \\ & T_j^L \leq f_T(T(t)), \ \ \forall t \in [0, t_f] \\ & X_{t_f}(T(t), t_f) \geq X_d \\ & M_{n,t_f}(T(t), t_f) \geq M_{n,d} \end{array} ,$$

As the temperature limits are semi-infinite in nature and have to be satisfied $\forall t \in [0, t_f]$, we carry out a finite reformulation as follows:

$$\begin{array}{ll} \underset{T(t),t_f}{\text{minimize}} & t_f \\ \text{subject to} & T^L \leq \min\limits_{t\in[0,t_f]} T(t) \\ & \max\limits_{t\in[0,t_f]} T(t) \leq T^U \\ & T_j^L \leq \min\limits_{t\in[0,t_f]} f_T(T(t)) \\ & X_{t_f}(T(t),t_f) \geq X_d \\ & M_{n,t_f}(T(t),t_f) \geq M_{n,d} \end{array},$$

and, noting that the minimum and maximum operators are functions of both $T(t)$ and t_f, adopt the notation:

$$\begin{array}{l} \underline{T}(T(t),t_f) = \min\limits_{t\in[0,t_f]} T(t) \\ \overline{T}(T(t),t_f) = \max\limits_{t\in[0,t_f]} T(t) \\ \underline{f}_T(T(t),t_f) = \min\limits_{t\in[0,t_f]} f_T(T(t)) \end{array},$$

which leads to:

$$\begin{array}{ll} \underset{T(t),t_f}{\text{minimize}} & t_f \\ \text{subject to} & T^L \leq \underline{T}(T(t),t_f) \\ & \overline{T}(T(t),t_f) \leq T^U \\ & T_j^L \leq \underline{f}_T(T(t),t_f) \\ & X_{t_f}(T(t),t_f) \geq X_d \\ & M_{n,t_f}(T(t),t_f) \geq M_{n,d} \end{array}. \tag{63}$$

We now have the task of parameterizing $T(t)$ so as to solve (63) as an RTO problem. The most basic parameterization, though not necessarily the most effective, is the approximation of $T(t)$ by a piecewise constant profile. For the case of just two pieces, this could be done as follows:

$$T(t) = \begin{cases} T_1, & 0 \leq t < t_f/2 \\ T_2, & t_f/2 \leq t \leq t_f \end{cases}.$$

As $T(t)$ is now fully parameterized by T_1, T_2, and t_f, this allows us to restate (63) as:

$$\begin{array}{ll} \underset{T_1,T_2,t_f}{\text{minimize}} & t_f \\ \text{subject to} & T^L \leq \underline{T}(T_1,T_2,t_f) \\ & \overline{T}(T_1,T_2,t_f) \leq T^U \\ & T_j^L \leq \underline{f}_T(T_1,T_2,t_f) \\ & X_{t_f}(T_1,T_2,t_f) \geq X_d \\ & M_{n,t_f}(T_1,T_2,t_f) \geq M_{n,d} \end{array}.$$

or, having defined $\mathbf{u} := [T_1\ T_2\ t_f]^T$, in standard RTO form as:

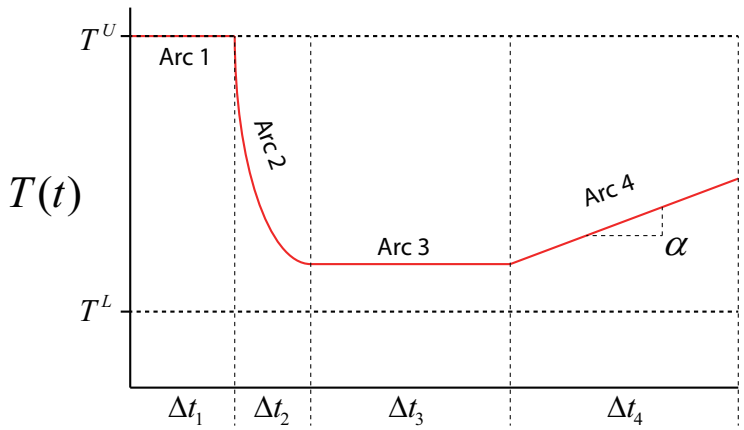

Figure 16: Solution model for the batch polymerization profile of [17].

$$
\begin{array}{lll}
\underset{\mathbf{u}}{\text{minimize}} & u_3 & \left.\begin{array}{l}\\\end{array}\right\} \phi(\mathbf{u}) \\
\text{subject to} & -\underline{f}_T(\mathbf{u}) + T_j^L \leq 0 & \left.\begin{array}{l} \\ \\ \\ \\ \\ \end{array}\right. \\
& -X_{t_f}(\mathbf{u}) + X_d \leq 0 & \\
& -M_{n,t_f}(\mathbf{u}) + M_{n,d} \leq 0 & \left\} \mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0} \right. \\
& T^L \leq u_1 \leq T^U & \\
& T^L \leq u_2 \leq T^U & \left\} \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \right. \\
& 0 \leq u_3 \leq \bar{t}_f &
\end{array}
\quad ,
$$

where the added box constraints imply the satisfaction of $T^L \leq T(t) \leq T^U$ and where $\bar{t}_f$ may simply be taken as some large number (or the final time of the initial batch, since optimization can only lower it).

An alternative parameterization is that of NCO ("necessary conditions of optimality") tracking [26, 25], which involves breaking $T(t)$ into a sequence of arcs based on engineering knowledge or prior model-based simulations [20]. The authors of [17] did this for Problem (62) by noting that the "best" batch operation for this process first involved keeping the batch at the maximum temperature, followed by cooling as quickly as possible, and then followed by keeping the temperature constant, before finishing with a temperature ramp (see [17] for the proper justifications). As shown in Figure 16, this essentially breaks the problem into four arcs, with the variables to optimize being the lengths of the arcs and the slope of the ramp at the end. Defining $\mathbf{u} := [\Delta t_1 \ \Delta t_2 \ \Delta t_3 \ \Delta t_4 \ \alpha]^T$, we may, without much effort, state the resulting RTO problem as:

$$
\begin{array}{rll}
\underset{\mathbf{u}}{\text{minimize}} & u_1 + u_2 + u_3 + u_4 & \left.\begin{array}{l}\\\end{array}\right\} \phi(\mathbf{u}) \\
\text{subject to} & -\underline{T}(\mathbf{u}) + T^L \leq 0 & \left.\begin{array}{l}\\\\\\\\\\\end{array}\right\} \mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0} \\
& \overline{T}(\mathbf{u}) - T^U \leq 0 & \\
& -\underline{f}_T(\mathbf{u}) + T_j^L \leq 0 & \\
& -X_{t_f}(\mathbf{u}) + X_d \leq 0 & \\
& -M_{n,t_f}(\mathbf{u}) + M_{n,d} \leq 0 & \\
& 0 \leq u_1 \leq \bar{t}_f & \left.\begin{array}{l}\\\\\\\\\\\end{array}\right\} \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \\
& 0 \leq u_2 \leq \bar{t}_f & \\
& 0 \leq u_3 \leq \bar{t}_f & \\
& 0 \leq u_4 \leq \bar{t}_f & \\
& 0 \leq u_5 \leq \overline{\alpha} &
\end{array}
,
$$

where both $\bar{t}_f$ and $\overline{\alpha}$ need to be sufficiently large.

Finally, one may use a polynomial parameterization as proposed in [18]. For a quadratic, this would yield:

$$T(t) = a_2 t^2 + a_1 t + a_0,$$

which lets $T(t)$ be entirely defined by a_2, a_1, and a_0. Setting $\mathbf{u} := [a_2\ a_1\ a_0\ t_f]^T$ leads to:

$$
\begin{array}{rll}
\underset{\mathbf{u}}{\text{minimize}} & u_4 & \left.\begin{array}{l}\\\end{array}\right\} \phi(\mathbf{u}) \\
\text{subject to} & -\underline{T}(\mathbf{u}) + T^L \leq 0 & \left.\begin{array}{l}\\\\\\\\\\\end{array}\right\} \mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0} \\
& \overline{T}(\mathbf{u}) - T^U \leq 0 & \\
& -\underline{f}_T(\mathbf{u}) + T_j^L \leq 0 & \\
& -X_{t_f}(\mathbf{u}) + X_d \leq 0 & \\
& -M_{n,t_f}(\mathbf{u}) + M_{n,d} \leq 0 & \\
& \underline{a}_2 \leq u_1 \leq \overline{a}_2 & \left.\begin{array}{l}\\\\\\\\\\\end{array}\right\} \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \\
& \underline{a}_1 \leq u_2 \leq \overline{a}_1 & \\
& \underline{a}_0 \leq u_3 \leq \overline{a}_0 & \\
& 0 \leq u_4 \leq \bar{t}_f &
\end{array}
,
$$

where the lower and upper bounds on the a coefficients need to be sufficiently far apart so as to allow the optimization sufficient room to produce a feasible solution and to optimize.

4.2.2 Simulation Example: Batch Reactor with Reversible Reaction

We consider the following reversible reaction taking place in a batch reactor, as proposed in [18]:

$$
\begin{array}{l}
A \underset{k_2}{\overset{k_1}{\leftrightarrows}} B, \quad r = k_1 C_A - k_2 C_B \\
k_1 = k_{1,0} e^{-\frac{E_1}{RT}}, k_2 = k_{2,0} e^{-\frac{E_2}{RT}}
\end{array}
, \tag{64}
$$

with the constants $k_{1,0} = 1.32 \cdot 10^7 \text{hr}^{-1}$, $k_{2,0} = 5.24 \cdot 10^{13} \text{hr}^{-1}$, $E_1 = 10,000 \frac{\text{cal}}{\text{g·mol}}$, and $E_2 = 20,000 \frac{\text{cal}}{\text{g·mol}}$, and the initial concentrations $C_{A,0} = 1.0$ and $C_{B,0} = 0$. The goal is to find the

optimal temperature profile $T(t)$ so as to maximize the conversion of A to B after 2.5 hours. A path constraint forces $T(t)$ to be between 20°C and 50°C at all times.

In this example, we use a simple two-parameter parameterization proposed in [18] to express the profile as:

$$\begin{array}{l} T(\tau) = 35 + 15w(\tau) \\ w(\tau) = a_1 + a_2(1 - 2\tau) \end{array} , \tag{65}$$

where $\tau \in [0, 1]$ is the normalized time and a_1 and a_2 are the optimization variables that parameterize $w(\tau)$, the dynamic portion of $T(\tau)$. By enforcing the (four) constraints:

$$-1 \leq a_1 \pm a_2 \leq 1$$

the author forces the resulting temperature profile to satisfy the upper and lower temperature limits.

Defining $\mathbf{u} := [a_1 \; a_2]^T$, the resulting RTO problem may be stated as:

$$\begin{array}{lll} \underset{\mathbf{u}}{\text{minimize}} & C_{A,t_f}(\mathbf{u}) & \left.\begin{array}{l} \\ \end{array}\right\} \phi_p(\mathbf{u}) \\ \text{subject to} & u_1 + u_2 - 1 \leq 0 & \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right. \\ & u_1 - u_2 - 1 \leq 0 & \\ & -u_1 - u_2 - 1 \leq 0 & \left\} \mathbf{G}(\mathbf{u}) \preceq \mathbf{0} \right. \\ & -u_1 + u_2 - 1 \leq 0 & \\ & -1 \leq u_1 \leq 1 & \left.\begin{array}{l} \\ \end{array}\right. \\ & -1 \leq u_2 \leq 1 & \left\} \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \right. \end{array} , \tag{66}$$

where we note that the final two constraints are redundant and implied by the others, but are nevertheless required for the standard RTO form and the solver. In this problem, conversion is maximized by minimizing the concentration of A at the final time, with the implicit function $C_{A,t_f}(\mathbf{u})$ denoting this value.

Noting that the problem is already well-scaled, we solve this problem with the SCFO solver as follows:

Initialization

To initialize the solver, we follow the methodology in [18] and conduct a 3^2 rotated factorial design of experiments over the reduced space given by:

$$-0.98 \leq u_1 \pm u_2 \leq 0.98,$$

where the reduction is done so as to ensure that all of the tested points will be feasible for the RTO problem once back-offs have been added to allow sufficient excitation. Specifically, the initial input set that follows from the design is:

$$\mathbf{U}_0 = \begin{bmatrix} -0.98 & 0 \\ -0.49 & 0.49 \\ 0.49 & 0.49 \\ 0.98 & 0 \\ 0.49 & -0.49 \\ 0 & -0.98 \\ -0.49 & -0.49 \\ 0 & 0 \\ 0 & 0.98 \end{bmatrix}.$$

Using the data obtained from these nine experiments, with the concentration values corrupted by additive noise from the distribution $\mathcal{N}(0, 10^{-4})$, a quadratic response surface model is fitted:

$$C_{A,t_f}(\mathbf{u}) \approx \beta_{11}u_1^2 + \beta_{22}u_2^2 + \beta_{12}u_1u_2 + \beta_1u_1 + \beta_2u_2 + \beta_0,$$

and is then optimized to find the predicted optimum, which is applied as the tenth point, $\mathbf{u}_9$:

$$\begin{aligned} \mathbf{u}_9 := \arg\underset{\mathbf{u}}{\text{minimize}} \quad & \beta_{11}u_1^2 + \beta_{22}u_2^2 + \beta_{12}u_1u_2 + \beta_1u_1 + \beta_2u_2 + \beta_0 \\ \text{subject to} \quad & u_1 + u_2 - 0.98 \leq 0 \\ & u_1 - u_2 - 0.98 \leq 0 \\ & -u_1 - u_2 - 0.98 \leq 0 \\ & -u_1 + u_2 - 0.98 \leq 0 \end{aligned}.$$

The SCFO solver is launched thereafter.

Noise Models

The noise model of the cost is assumed known as $\mathcal{N}(0, 10^{-4})$.

Lipschitz and Quadratic Bound Constants

We use the same procedure as for the Williams-Otto example earlier (Section 4.1.2), and employ the response surface model to calculate the minimal and maximal derivatives over the constrained input space. In this case, this does not require gridding and is easily done via linear programming. For example, to obtain the minimal Lipschitz constant estimate with respect to u_1, we simply solve the following:

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & 2\beta_{11}u_1 + \beta_{12}u_2 + \beta_1 \\ \text{subject to} \quad & u_1 + u_2 - 1 \leq 0 \\ & u_1 - u_2 - 1 \leq 0 \\ & -u_1 - u_2 - 1 \leq 0 \\ & -u_1 + u_2 - 1 \leq 0 \end{aligned},$$

and then add conservatism by either doubling or halving, as described before.

As the second derivatives of the quadratic model are constant, with the Hessian

$$\mathbf{H} = \begin{bmatrix} 2\beta_{11} & \beta_{12} \\ \beta_{12} & 2\beta_{22} \end{bmatrix},$$

a certain degree of robustness against the uncertainty of this model is added by defining the M constants as follows:

$$\underline{M}_{ij} := H_{ij} - |H_{ij}|, \;\; \overline{M}_{ij} := H_{ij} + |H_{ij}|,$$

which is, admittedly, a heuristic choice, but one that has worked well for us.

The Lipschitz constants for the certain constraints are easily defined as:

$$\underline{\mathbf{K}} := \begin{bmatrix} 0.99 & 0.99 \\ 0.99 & -1.01 \\ -1.01 & -1.01 \\ -1.01 & 0.99 \end{bmatrix}, \;\; \overline{\mathbf{K}} := \begin{bmatrix} 1.01 & 1.01 \\ 1.01 & -0.99 \\ -0.99 & -0.99 \\ -0.99 & 1.01 \end{bmatrix}.$$

Lower Scaling Bounds

It is easily shown that the lowest value achievable by all of the four constraints is -2, and so choose $\underline{\mathbf{G}} := [-2 \;\; -2 \;\; -2 \;\; -2]$.

Global Minimum Cost and Convergence Tolerance

We do not attempt to use this option here and so set $\underline{\phi} := 0$ (which is very unlikely to be reached, as this corresponds to full conversion) and $\epsilon_\phi := 0$.

Maximum Input Step Size

$\mathbf{\Delta u}_{max}$ is defined as [0.20 0.20], which corresponds to 10% of the input domain for both inputs.

The results are presented in Figure 17, where one sees that one of the initial experiments is able to hit the optimum almost exactly, after which the SCFO solver takes this point and uses it as the reference, staying in the neighborhood of the optimum despite significant measurement errors due to noise.

4.3 Iterative Controller Tuning

In certain applications, a controller is tasked with the job of following the same reference trajectory from one run/batch/cycle to another [29]. Since the way in which the controller is tuned (i.e., what controller parameters, $\boldsymbol{\rho}$, are specified) affects its performance (e.g., its ability to track the trajectory), it is possible to adapt the controller parameters iteratively from run to run so as to find the locally optimal set of parameters (Figure 18).

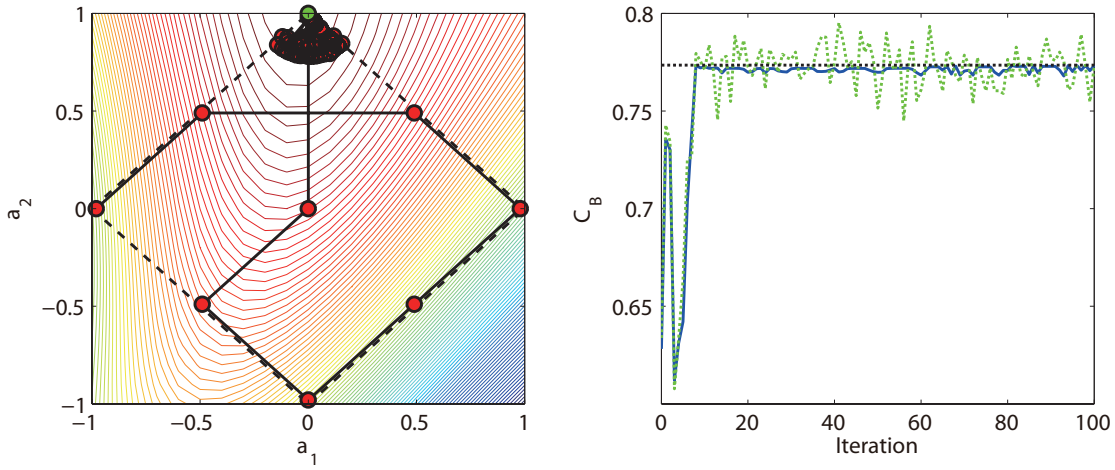

Figure 17: The SCFO solver applied to a parameterized dynamic optimization problem, following a design-of-experiments initialization. The dashed lines on the left-hand-side figure denote the inequality constraints that fix the optimization domain.

4.3.1 Formulation as an RTO Problem

Denoting by J_k the observed performance of the controller at run (iteration) k, one may solve this as an RTO problem by making the following *repeatability assumption* [4]:

$$J_k = J(\boldsymbol{\rho}_k) + \delta_k, \tag{67}$$

where J is an unknown but deterministic relation between the controller parameters $\boldsymbol{\rho}$ and the performance, while δ_k is a stochastic "noise" element. This assumption is in line with what would normally be expected – applying the same controller for the same trajectory in two different runs should yield similar or close to identical performance, with the degree of similarity inversely proportion to the magnitude of the noise term.

With this assumption and the definition $\mathbf{u} := \boldsymbol{\rho}$, the observed performance may be optimized by simply solving:

$$\begin{array}{lll} \underset{\mathbf{u}}{\text{minimize}} & J(\mathbf{u}) & \left.\begin{array}{l}\\\\\\\end{array}\right\} \phi_p(\mathbf{u}) \\ \text{subject to} & \mathbf{G}_{out}(\mathbf{u}) \preceq \mathbf{0} & \left.\begin{array}{l}\\\\\end{array}\right\} \mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0} \\ & \mathbf{G}_{des}(\mathbf{u}) \preceq \mathbf{0} & \left.\begin{array}{l}\\\end{array}\right\} \mathbf{G}(\mathbf{u}) \preceq \mathbf{0} \\ & \boldsymbol{\rho}^L \preceq \mathbf{u} \preceq \boldsymbol{\rho}^U & \left.\begin{array}{l}\\\end{array}\right\} \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \end{array}, \tag{68}$$

where $\mathbf{G}_{out}$ denotes potential output constraints, $\mathbf{G}_{des}$ denotes design constraints that are controlled directly by the user (e.g., a constraint on controller stability), and $\boldsymbol{\rho}^L$ and $\boldsymbol{\rho}^U$ define the domain over which the parameters will be tuned.

A potential difficulty in this problem is the abstract "noise" term δ_k, for which obtaining an accurate PDF may be difficult, with crude estimations having to suffice. In some cases, the nature of δ_k may also vary with parameter values (thereby violating the repeatability assumption),

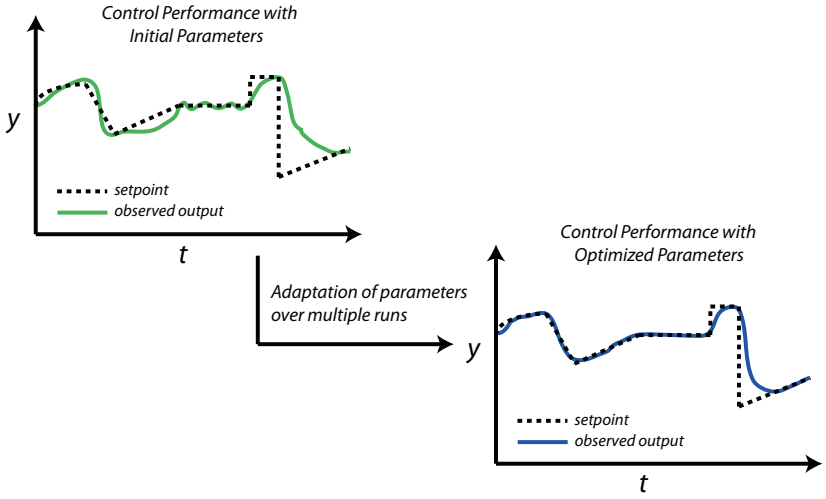

Figure 18: The main idea of iterative controller tuning.

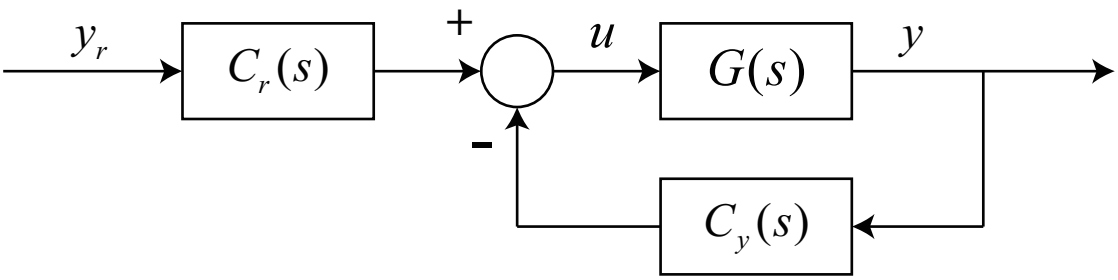

Figure 19: The control scheme considered in the example problem.

which may mean that the PDF assumed for one set of parameters may not be a valid PDF for another.

Another danger is that closed-loop stability cannot be guaranteed and is only encouraged – by enforcing that the performance improve continuously and that any important safety constraints be met, the SCFO tends to indirectly avoid the kind of instability that would make for dangerous operation.

4.3.2 Simulation Example: PID Tuning for a Step Setpoint Change

We consider the three-parameter PID controller scheme given in Figure 19, for which similar studies have been carried out in the context of iterative-feedback [22] and extremum-seeking-based [21] tuning, and where:

$$C_r(s) = K_p \left(1 + \frac{1}{T_i s}\right)$$

$$C_y(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s\right),$$

with $\boldsymbol{\rho} = [K_p \;\; T_i \;\; T_d]^T$ the three parameters to be tuned.

The system considered in this example is:

$$G(s) = \frac{3}{s^3 + 2s^2 + s + 2},$$

with the goal of minimizing the tracking error of the response over a specified time horizon:

$$J(\boldsymbol{\rho}) := \int_5^{40} \left[y_r(t) - y(t, \boldsymbol{\rho})\right]^2 dt,$$

when a unit step is applied to change the output setpoint from 0 to 1, and where we note that the observed response $y(t, \boldsymbol{\rho})$ is essentially a black-box function of the controller parameters. Adopting the proposition in [22], a "mask" of 5 time units is applied so as not to judge controller performance during the very beginning of the response where large tracking errors will be inevitable due to the abrupt nature of the setpoint change.

Additionally, it is desired that the output never overshoot the setpoint by more than 0.1, with the constraint

$$y(t, \boldsymbol{\rho}) \leq 1.1, \;\; \forall t \in [0, 40]$$

enforced as a *hard output constraint*. By applying the repeatibility assumption on this constraint, we may define:

$$y_{max}(\boldsymbol{\rho}) := \max_{t \in [0,40]} y(t, \boldsymbol{\rho}),$$

and then write this constraint in finite form as:

$$y_{max}(\boldsymbol{\rho}) \leq 1.1.$$

Based on experience with similar problems [22, 21], we expect T_i and T_d to vary more (about a magnitude more) than K_p, and so we choose to apply the scaling $T_i := T_i/10$, $T_d := T_d/10$. Since the exact lower and upper limits on the three parameters are unknown (apart from the fact that T_i and T_d should be positive), *adaptive limits* are used, with:

$$\begin{array}{l} K_{p,k} - 0.5 \leq K_p \leq K_{p,k} + 0.5 \\ \max(0, T_{i,k} - 5) \leq T_i \leq T_{i,k} + 5 \\ \max(0, T_{d,k} - 5) \leq T_d \leq T_{d,k} + 5 \end{array},$$

which lead to an input domain of no more than 1 for all parameters at a given iteration k post scaling (the limits above being unscaled for T_i and T_d). Note that these domains are artificial

and are almost never limiting, as they always move with the parameters – however, they provide a convenient and well-scaled choice for the solver, which requires the limits.

With $\mathbf{u} := \boldsymbol{\rho}$, we may now define the RTO problem as:

$$\begin{array}{lll} \underset{\mathbf{u}}{\text{minimize}} & \displaystyle\int_5^{40} \left[y_r(t) - y(t,\mathbf{u})\right]^2 dt & \left.\begin{array}{l}\\\end{array}\right\} \phi_p(\mathbf{u}) \\ \text{subject to} & y_{max}(\mathbf{u}) - 1.1 \leq 0 & \left.\begin{array}{l}\\\end{array}\right\} \mathbf{G}_p(\mathbf{u}) \preceq \mathbf{0} \\ & K_p^L(k) \leq u_1 \leq K_p^U(k) & \\ & T_i^L(k) \leq u_2 \leq T_i^U(k) & \left.\begin{array}{l}\\\\\\\end{array}\right\} \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \\ & T_d^L(k) \leq u_3 \leq T_d^U(k) & \end{array},$$

The cost is scaled by division by its initial value, and the problem is solved with the SCFO solver as follows:

Initialization

An initial parameter set of $\mathbf{u}_0 := [2\ 10\ 2]^T$ (pre-scaling) is chosen as this achieves reasonable (though suboptimal) performance while satisfying the output constraint by a significant margin. The smart perturbation algorithm described in Section 4.1.2 is then applied to generate $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$ with $\Delta\mathbf{u}_{max} := [0.1\ 0.1\ 0.1]$ post-scaling. A noise of $\delta_k \sim \mathcal{N}(0, 2.5 \cdot 10^{-4})$ is added to the scaled performance metric following each evaluation, with a noise of $\mathcal{N}(0, 10^{-4})$ added to the output constraint measurement.

Nominal RTO Target

The nominal RTO target, $\mathbf{u}_{k+1}^*$, is chosen by implementing a gradient-descent step scaled by the Hessian of the quadratic model fitted to the data (see the discussion on the choice of quadratic bound constants below):

$$\mathbf{u}_{k+1}^* = \mathbf{u}_k - \mathbf{H}^\dagger \nabla\hat{\phi}_p(\mathbf{u}_k),$$

where the estimate $\nabla\hat{\phi}_p(\mathbf{u}_k)$ is obtained as the gradient of the fitted quadratic model. A pseudoinverse is used in case the Hessian happens to be very badly conditioned or not invertible. Although a step to ensure that the Hessian is positive definite should normally be carried out, we do not do so here, believing the RTO target as given above to be a sufficiently good guide.

Noise Models

We assume that the noise models for the cost and constraint are known in this example.

Lipschitz and Quadratic Bound Constants

We assume to work in a model-free setting, and so the Lipschitz and quadratic bound constants must be chosen by heuristic means. A heuristic that we propose here, and that has, so far, worked well for us, is the following.

To obtain a (very conservative) estimate of the Lipschitz constants for the cost, take the initial data set and fit it with a linear model, the coefficients of which are then the initial gradient estimate:

$$\begin{bmatrix} \nabla\hat{\phi}_p(\mathbf{u}_0) \\ a_0 \end{bmatrix} = \begin{bmatrix} \mathbf{u}_0^T & 1 \\ \mathbf{u}_1^T & 1 \\ \mathbf{u}_2^T & 1 \\ \mathbf{u}_3^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} \hat{\phi}_p(\mathbf{u}_0) \\ \hat{\phi}_p(\mathbf{u}_1) \\ \hat{\phi}_p(\mathbf{u}_2) \\ \hat{\phi}_p(\mathbf{u}_3) \end{bmatrix}.$$

The Lipschitz constants are then defined as:

$$\overline{\kappa}_{\phi,i}, \underline{\kappa}_{\phi,i} := \pm 2 \left\| \nabla\hat{\phi}_p(\mathbf{u}_0) \right\|_\infty, \quad i = 1, ..., n_u.$$

For the quadratic bounds, an adaptive choice is used, with the basic idea of using the Hessian of the quadratic model that is fit to all of the measurements at every new iteration as a guide. Since not enough measurements are available initially to estimate a full quadratic, the following scheme is adopted:

- When the number of measurements is less than 7, define $\mathbf{H}$ as a diagonal matrix with:

$$H_{ii} := 5 \left\| \nabla\hat{\phi}_p(\mathbf{u}_0) \right\|_\infty, \quad i = 1, ..., n_u.$$

- When the number of measurements is greater than 7 but less than 10, fit a quadratic model without interaction terms for all of the data:

$$\phi_p(\mathbf{u}) \approx \beta_{11}u_1^2 + \beta_{22}u_2^2 + \beta_{33}u_3^2 + \beta_1 u_1 + \beta_2 u_2 + \beta_3 u_3 + \beta_0,$$

  with $\mathbf{H}$ the (diagonal) Hessian of this model. Apply the following trimming if necessary (as a safeguard):

$$\begin{array}{l} H_{ij} > 5 \left\| \nabla\hat{\phi}_p(\mathbf{u}_0) \right\|_\infty \rightarrow H_{ij} := 5 \left\| \nabla\hat{\phi}_p(\mathbf{u}_0) \right\|_\infty \\ H_{ij} < -5 \left\| \nabla\hat{\phi}_p(\mathbf{u}_0) \right\|_\infty \rightarrow H_{ij} := -5 \left\| \nabla\hat{\phi}_p(\mathbf{u}_0) \right\|_\infty \end{array}.$$

- When the number of measurements is 10 or more, fit a full quadratic model to all of the data:

$$\phi_p(\mathbf{u}) \approx \beta_{11}u_1^2 + \beta_{22}u_2^2 + \beta_{33}u_3^2 + \beta_{12}u_1u_2 + \beta_{13}u_1u_3 + \beta_{23}u_2u_3 + \beta_1 u_1 + \beta_2 u_2 + \beta_3 u_3 + \beta_0$$

  with $\mathbf{H}$ the Hessian of this model. Apply the same trimming as above, if needed.

Due to model uncertainty, the actual M constants are defined as:

$$\overline{M}_{ij}, \underline{M}_{ij} := H_{ij} \pm \sigma_s |H_{ij}|,$$

where σ_s is a safety factor that is initialized at 1 and then updated iteratively as follows for all iterations where sufficient excitation is not enforced (when $\delta_{exit} = 0$):

- If $J(\mathbf{u}_k) - 4\sigma_\delta \geq \min_{i=0,...,k-1} J(\mathbf{u}_i)$, then set $\sigma_s := \sigma_s + 1$,

- otherwise, set $\sigma_s := \sigma_s - 0.5$, with $\sigma_s < 0 \rightarrow \sigma_s := 0$,

where σ_δ is the standard deviation of the noise term δ_k.

Such an update law essentially acts, in a very informal sense, as a control scheme, so that the safety factor is augmented whenever a decrease in the cost is not achieved (which, in the presence of good gradient estimates, would be due to a poor estimate of the quadratic bound constants), and decreased when a cost decrease is noted (so as to reduce potential conservatism and speed up convergence). Having the decrease at half the value of the increase (0.5 compared to 1) is a heuristic choice and is intended to err on the side of conservatism, thereby encouraging a larger, and safer, quadratic upper bound.

For the Lipschitz constants of the output constraint $g_p(\mathbf{u}) := y_{max}(\mathbf{u}) - 1.1$, the same initial procedure is used:

$$\overline{\kappa}_i^p, \underline{\kappa}_i^p := \pm 2 \left\| \nabla \hat{g}_p(\mathbf{u}_0) \right\|_\infty , \quad i = 1, ..., n_u,$$

Additional safeguards should also be taken to account for the possibility of $\left\| \nabla \hat{g}_p(\mathbf{u}_0) \right\|_\infty \approx 0$. If this is the case (which would suggest that the initialization takes place in a zero-sensitivity region for the constraint), a brute non-zero estimate for the Lipschitz constants may be:

$$\overline{\kappa}_i^p, \underline{\kappa}_i^p := \pm \frac{-\underline{g}_p}{u_i^U - u_i^L}, \quad i = 1, ..., n_u,$$

where $\underline{g}_p$ is simply an element of $\underline{\mathbf{G}}_p$.

Lower Scaling Bounds

As the maximum output value should be no less than 1 if the controller does its job, we set $\underline{G}_p := -0.1$.

Global Minimum Cost and Convergence Tolerance

Since the criterion is to minimize the tracking error, we can do no better than to lower it to $\underline{\phi} := 0$, and should not attempt to do any better. In this example, we will assume that meeting this minimum value with a tolerance of $\epsilon_\phi := 0.1$ is sufficient.

Concavity Matrix

The concavity matrix is defined as $\mathbf{C} := [0\ 0\ 0]$, as there is no basis for any concave relationships between the output constraints and the parameters.

Constraint Relaxations

No relaxation is allowed for the hard output constraint, and so we set $\mathbf{d} := \mathbf{d}_T := 0$.

The results for this problem are presented in Figure 20, where we see that the solver is able to obtain a significantly better controller fairly quickly (less than 20 iterations) without violating

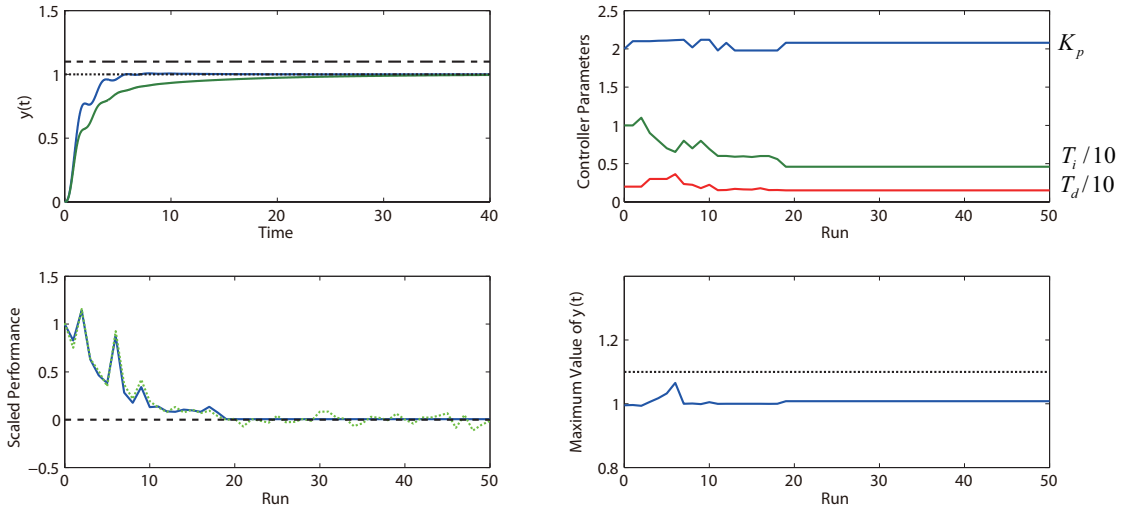

Figure 20: Application of the SCFO solver to the PID tuning problem.

the output constraint. After reaching a sufficiently optimal performance, no further adaptation is done.

4.3.3 Experimental Examples: MPC and General Fixed-Order Controller Tuning (Description Only)

Some promising results have been obtained in two different experimental settings, but as they are currently in the process of being published [6], we can unfortunately only provide a qualitative description of the problems.

The first problem involves tuning a model-predictive controller (MPC) for the setup previously described in [10], where a stirred water tank is heated/cooled by the water in the surrounding jacket, with the MPC setting the temperature of the jacket inlet so as to track a setpoint profile for the water inside the tank (for a simplified schematic, see Figure 21).

As is required for the iterative controller tuning problem, the temperature setpoint profile is kept the same from batch to batch, with the performance criterion chosen as the minimization of the squared tracking error over the batch time t_b:

$$J(\boldsymbol{\rho}) := \int_0^{t_b} \left[T_r^{set}(t) - T_r(t, \boldsymbol{\rho})\right]^2 dt,$$

where $\boldsymbol{\rho}$ represent the tuning parameters of the MPC and may be any of the following: the weights of the MPC penalty function, the prediction and control horizons, or the bias update filter gain [4]. Casting this as an RTO problem and applying the SCFO solver is straightforward and may be done in the same manner as for the simulation example above.

In another tuning problem, a much faster mechanical torsional system was considered (Figure 22), where the motor torque was used to control the angular position of the top disk in a three-disk setup. In contrast to the batch reactor case, what constituted a single iteration here was not a single "batch" but a period in a repetitive setpoint trajectory, with the latter given by:

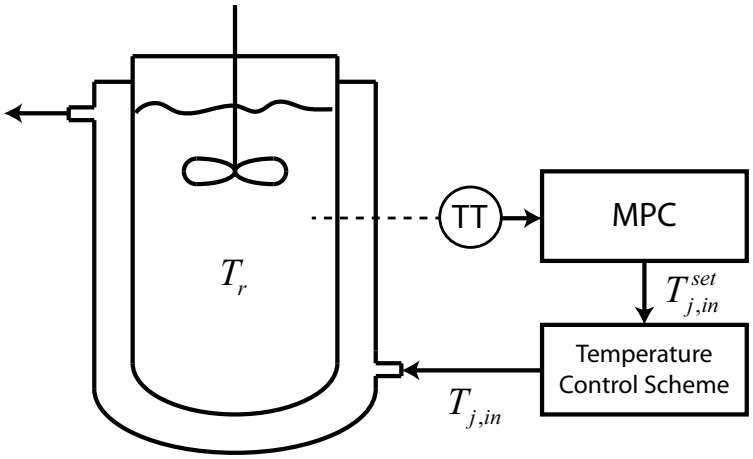

Figure 21: Qualitative schematic of MPC temperature control in a jacketed stirred tank.

$$y_r(t, A) = A \text{ sign}\left(\sin\frac{\pi t}{6}\right),$$

i.e., a square wave with a period of 12 seconds and an amplitude of A.

The performance metric in this case was chosen as a more general weighted combination of tracking error, controller aggressiveness, and the smoothness of the output profile:

$$\begin{aligned} J_k := & \sum_{T=200k}^{200(k+1)-1} \left[y_r(0.06T, A) - y(0.06T, \boldsymbol{\rho}_k)\right]^2 \\ & +\lambda_1 \sum_{T=200k}^{200(k+1)-1} \left[u(0.06T, \boldsymbol{\rho}_k) - u(0.06T - 0.06, \boldsymbol{\rho}_k)\right]^2 \\ & +\lambda_2 \sum_{T=200k}^{200(k+1)-1} \left[y(0.06T, \boldsymbol{\rho}_k) - y(0.06T - 0.06, \boldsymbol{\rho}_k)\right]^2 \end{aligned},$$

with $\lambda > 0$ denoting the weights and T the discrete sample counter.

The controller used in this study was a discrete fixed-order controller of the form:

$$G(\boldsymbol{\rho}) = \frac{\rho_1 z^2 + \rho_2 z + \rho_3}{z^2 + \rho_4 z + \rho_5},$$

with the coefficients of the controller numerator and denominator being the tuned parameters.

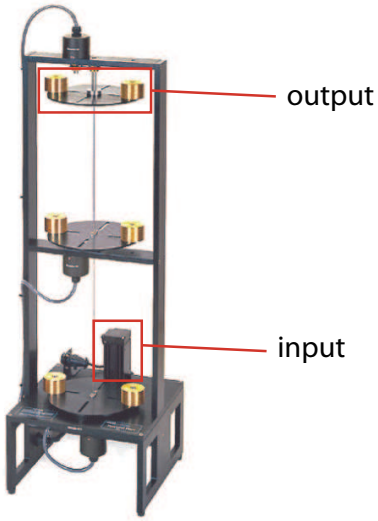

Figure 22: The torsional system where the motor torque is used to control the angular position of the top disk.

4.4 Optimization by Response-Surface Methods

A very large number of practical optimization problems come without a model and are solved by response-surface methods (we refer the user to [23] and the numerous examples therein), where a certain set of experiments is carried out so as to construct an approximate first- or second-order data-driven model, which is then optimized to find the best design or operating conditions for the problem at hand. Such problems require, in general, no special reformulation to fit into our RTO framework, as they are innately posed as RTO problems with clearly defined input variables $\mathbf{u}$, a criterion to be optimized ϕ_p, and the constraints $\mathbf{G}_p$ and $\mathbf{G}$, as well as the design space given by $\mathbf{u}^L$ and $\mathbf{u}^U$. As such, they may be solved directly using the SCFO solver, with perhaps the only differences being in how the data set is initialized.

We propose two initialization candidates here and use the previously presented example of Section 4.2.2 as an illustration.

4.4.1 Design of Experiments Followed by SCFO

As already illustrated in Section 4.2.2, one may use the SCFO solver after an initial design-of-experiments phase. This is particularly nice with respect to the choice of Lipschitz and quadratic bound constants, as the response-surface model may be used to easily approximate these values. The results in Figure 17 show that the SCFO solver can successfully act as the follow-up step.

4.4.2 Initialization at the Center Followed by SCFO

An alternative is to skip the response-surface modeling step and to launch the SCFO solver straight away, following a basic initialization from the very center of the design space.

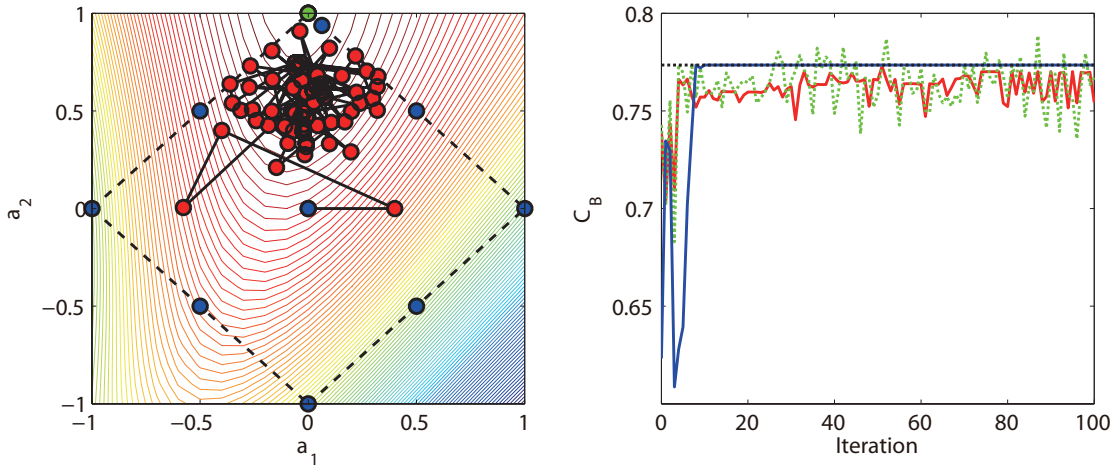

Figure 23: The SCFO solver applied to a parameterized dynamic optimization problem directly after a basic initialization from the center of the input space (skipping the response-surface model building step). A comparison is made with the "classical" response-surface optimization scheme, with the blue dots on the left denoting the experiments carried out to construct the response-surface model, followed by an additional experiment that corresponds to the predicted optimum. The figure on the right shows the performance of the SCFO (in red) compared to the classical scheme, where the first ten iterations are used for model-building and optimization and the remaining 90 are kept at the best point found by this method.

Here, we define $\mathbf{u}_0$ as $[0 \quad 0]^T$ and perturb by the maximal allowable amount $\Delta\mathbf{u}_{max}$ via the "smart" perturbation scheme outlined in the simulation example of Section 4.1.2. Once $\mathbf{u}_1$ and $\mathbf{u}_2$ are obtained this way, we define the Lipschitz constants and the (adapted) quadratic bounds by the same heuristic rules as in Section 4.3.2. We note that $\Delta\mathbf{u}_{max}$ is chosen as 20% of the feasible domain in this example (as opposed to 10%).

There are two immediately obvious disadvantages to this alternative. First, since the SCFO solver is designed to converge to the local optimum in the neighborhood of the best available data point that is provided to it, there is a chance that very good points will be missed by starting at the center, which is less likely to occur when a response-surface model is built first, as the latter does a sort of global sampling of the design space. Additionally, the rules for choosing the Lipschitz and quadratic bound constants are "more heuristic" in nature than those used when a data-driven model is built first.

However, a very powerful practical advantage of this alternative is in that, for cases when the local optimum in the neighborhood of the center point is a very good one, the optimality losses attained during the suboptimal iterations are significantly less, as particularly "bad" experiments that would be done to construct the response-surface model are avoided. Figure 23 illustrates this, where it is seen that, although the response-surface methodology finds something very close to the optimum, the SCFO solver initialized from the center is able to achieve comparable results without having to carry out particularly suboptimal experiments.

4.5 Numerical Optimization with Expensive Function Evaluations

While the SCFO solver could, in principle, be applied to numerical optimization problems as well as RTO ones, in the vast majority of cases this is not practical, as the full chain of subroutines used in the solver is quite time-consuming and itself involves solving many numerical optimization problems. As such, while the iteration trajectory might be more efficient than that achieved by many numerical solvers (since the satisfaction of constraints and cost improvement is not ensured by a line search), the actual effort between iterations far overwhelms the simple calculations used in most numerical schemes.

However, there do exist purely numerical optimization problems where function evaluations are expensive (i.e., time consuming) and where typical optimization routines like, for example, calculating a gradient by finite difference perturbations and then doing a line search in the gradient descent direction, are simply too slow. Two examples of problems where such issues might arise are:

- problems where evaluating a function requires the lengthy simulation of a large/complex model,
- multi-level optimization problems, where evaluating a cost or constraint function involves solving another (potentially difficult) optimization problem.

A popular class of methods for solving such problems belongs to the derivative-free branch of numerical optimization [12], which includes methods like the direct search, the simplex, and trust-region algorithms. However, the SCFO solver may also be applied to successfully solve such problems as they are, in many ways, identical to RTO problems in that the functions, though known analytically, are expensive to evaluate in the same way that experiments are expensive to run.

4.5.1 Example: Bi-Level Optimization Problem

We consider the following (purely contrived) bi-level optimization problem:

$$\begin{array}{lll} \underset{u_1,u_2}{\text{minimize}} & f(u_1)(5-u_2^2) & \left.\begin{array}{l}\\\end{array}\right\} \phi_p(\mathbf{u}) \\ \text{subject to} & -2 \le u_1 \le 2 & \left.\begin{array}{l}\\\\\end{array}\right\} \mathbf{u}^L \preceq \mathbf{u} \preceq \mathbf{u}^U \\ & -2 \le u_2 \le 2 & \end{array} ,$$

where $f(u_1)$ denotes the globally minimal cost value of the following nonconvex problem:

$$\begin{array}{ll} \underset{x_1,\ldots,x_{10}}{\text{minimize}} & \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}(u_1)^T\mathbf{x} \\ \text{subject to} & \mathbf{A}_l\mathbf{x} \preceq \mathbf{b}_l \\ & x_i \in [0,1], \;\; i = 1,\ldots,10 \end{array} ,$$

where $\mathbf{A}_l$ and $\mathbf{b}_l$ are defined as:

$$\mathbf{A}_l = \begin{bmatrix} 2 & -6 & -1 & 0 & -3 & -3 & -2 & -6 & -2 & -2 \\ 6 & -5 & 8 & -3 & 0 & 1 & 3 & 8 & 9 & -3 \\ -5 & 6 & 5 & 3 & 8 & -8 & 9 & 2 & 0 & -9 \\ 9 & 5 & 0 & -9 & 1 & -8 & 3 & -9 & -9 & -3 \\ -8 & 7 & -4 & -5 & -9 & 1 & -7 & -1 & 3 & -2 \end{bmatrix}, \quad \mathbf{b}_l = \begin{bmatrix} -4 \\ 22 \\ -6 \\ -23 \\ -12 \end{bmatrix},$$

and where $\mathbf{Q} = -50\mathbf{I}$, while the linear portion varies as a function of u_1:

$$\mathbf{c}(u_1) = [48u_1 \;\; 42u_1^2 \;\; -48u_1 \;\; \frac{45}{3+u_1} \;\; 44u_1 \;\; -41u_1 \;\; 47u_1^2 \;\; 42-20u_1 \;\; 45\sqrt{3+u_1} \;\; 46u_1]^T.$$

We note that this problem, apart from the dependence of $\mathbf{c}$ on u_1, is identical to Test Problem 2.6 from [15].

Various global optimization methods may be used to solve the problem to evaluate $f(u_1)$ – here, we use the reverse-convex programming method utilized to solve a very similar problem in [3].

As ϕ_p has values on the magnitude of 10^3, we scale the cost as $\phi_p(\mathbf{u}) := \phi_p(\mathbf{u})/1000$. The problem is solved using the SCFO solver by applying the following configurations:

Initialization

The initial point is chosen as $\mathbf{u}_0 = [0 \;\; 0]^T$, and the "smart" initialization scheme given in Section 4.1.2, with $\Delta\mathbf{u}_{max} := [0.4 \; 0.4]$, is applied to generate $\mathbf{u}_1$ and $\mathbf{u}_2$, after which the SCFO solver is launched.

Nominal RTO Target

The nominal RTO target, $\mathbf{u}^*_{k+1}$, is chosen by fitting a quadratic model to all of the data and optimizing this model over the relevant input space.

Lipschitz and Quadratic Bound Constants

The Lipschitz and (adapted) quadratic bound constants are chosen by the heuristic laws used in Section 4.3.2.

Global Minimum Cost and Convergence Tolerance

We do not know *a priori* how low the scaled cost function can go, and so set $\underline{\phi} := -5$ as an estimated desired (and likely unreachable) target. For simplicity, we do not attempt to employ the convergence option here and so set $\epsilon_\phi := 0$.

The results are presented in Figure 24 and show that the solver is able to converge fairly quickly to the neighborhood of the local optimum and to remain there.

A potentially interesting extension of this example is the case where real-time constraints do not allow the global optimization problem to be solved in the time allotted. As many global

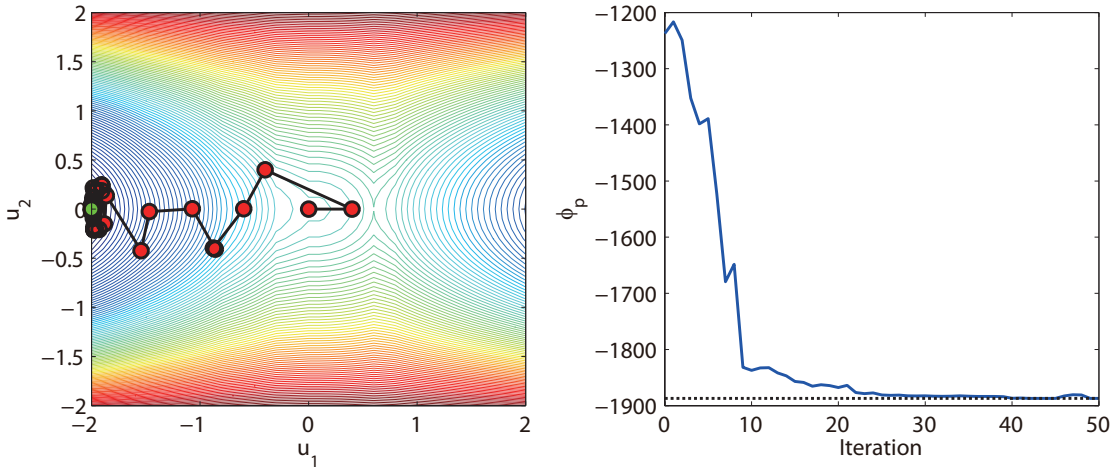

Figure 24: The SCFO solver applied to a numerical bi-level optimization problem.

optimization solvers – the one used here nonexempt – put upper and lower bounds on the cost during the solution process, one could terminate the solver early to yield the bounds obtained in the time allotted, rather than the exact cost value, which may require too much time to compute. In the SCFO framework, this essentially becomes identical to uniformly distributed measurement noise, and could therefore be treated as such – thereby allowing the user to work with very approximate solutions rather than exact global ones, and potentially achieving significant gains in speed.

5 Troubleshooting

This section attempts to address the main (and, when possible, minor) issues that may be encountered while trying to run the SCFO solver. Unfortunately, we can only address issues that we ourselves are aware of, and there will undoubtedly be bugs and other problems that we have not foreseen. If you encounter any problems that are not discussed here, please do not hesitate to let us know, as we will do our best to fix these.

5.1 Errors While Trying to Run `SCFO.m/SCFO_oct.m`

If you encounter a MATLAB/Octave error while trying to launch the `SCFO.m/SCFO_oct.m` file and are not sure what is causing it, there is a good chance that it is due to dimensionality issues in the inputs of `SCFO.m/SCFO_oct.m`. Please have a look at the list in Section 2.2 and make sure that all the dimensions of the file inputs are correct, that the domains used are not in violation with what is specified (e.g., a negative number for a setting that only accepts positive ones), and that you have not forgotten any inputs. Future versions will likely automate this check for you, but please do the verification yourself for the time being.

If you receive the error

```
Error:  Provided data should include at least one strictly feasible point.
```

then please make sure that your data has at least one point for which you are certain that all of the constraints (both $\mathbf{G}_p$ and $\mathbf{G}$, as well as the box constraints) are met robustly while accounting for back-off due to the sufficient-excitation parameter $\underline{\delta}_e$. Some common cases where this could be an issue are:

- Large variance specified for the noise in the constraint measurements/estimates, with the constraint close to active for all points in the data set. Here, the PDF of the corresponding noise would be very wide and have a large variance, thereby making it impossible to say that the constraint is really met with sufficiently high confidence. Either lower the variance of the PDF (if the initial choice is too conservative) or attempt to find data points that are "deeper" in the feasible set via additional experiments.
- Very conservative Lipschitz constants, thereby creating large back-offs in the RTO problem (see Section 3.6.1 for why this would occur) and making it more likely that the provided data does not possess a point satisfying these back-offs. In this case, try to lower the Lipschitz constants if this is justifiable.

We also note that the optimization solvers used in the subroutines of the SCFO solvers are not perfect and may fail from time to time. The current workarounds for these issues are heuristic and generally perform acceptably, although there may be issues that we are unaware of. In future versions, we will attempt to address these problems more rigorously and to robustify the optimization subroutines.

5.2 Very Slow (Or No) Improvement Observed

It may occur that you run the SCFO solver only to find that you are not doing any better than you were without it – we do not promise success for all cases, although we often expect it.

Failure to see improvement may occur due to several reasons:

- You are already at, or very close to, a local minimum (this is the good case!).
- The measurement noise is very large, therefore making the task of gradient estimation very difficult. When the solver is not able to obtain tight gradient bounds due to significant noise, it often tends to take smaller steps for robustness reasons. Large measurement noise in the constraints also slows down the algorithm since it may be difficult for it to guarantee robust feasibility in this case for large input steps.
- Either the Lipschitz or quadratic bound constants, or both, are overly conservative, thereby forcing the solver to take very small steps so as to ensure that the constraints are not violated and that the cost is lower at the next iteration. As mentioned in Section 2.4.2, it is good practice to make these bounds as tight as allowable, and doing this will speed up convergence. If the values must be conservative to be valid, then it may just be that one has to accept slow convergence.
- The reference input point is very close to one of the unknown inequality constraints $\mathbf{G}_p$, which forces the algorithm to take very small steps so as to guarantee that these constraints are not violated.

5.3 Important Safety Constraints are Violated

Safety constraints that are defined in the RTO problem in the set $\mathbf{G}_p$ but are violated during operation (when this is not desired/acceptable) suggest one of the following:

- The Lipschitz constants are poorly chosen and should be more conservative (the lower constant is too high or the upper constant is too low, or both).
- The noise PDFs for the constraints are not correct and predict the noise/error to be smaller than it really is.
- A concavity relaxation is made (Section 3.8.2) when it is not valid.

5.4 It Takes a Very Long Time to Get an Answer

For large data sets (e.g., more than 200 data points), it may take a very long time (i.e., more than a few minutes, or even hours) for the SCFO solver to produce an answer. This may be caused by several factors, and will generally be worse when the RTO problem has a lot of uncertain constraint functions and a greater number of inputs. Sometimes this may also occur due to issues with the optimization subroutines. The only advice that we have for you at the moment is to use the fast version of the solver if/when speed becomes an issue.

Future versions will attempt to treat this more elegantly by incorporating real-time constraints, thereby allowing for the user to control how long the SCFO solver takes, with certain subroutines being sacrificed – in order of importance – to meet the time constraints imposed by the user. Naturally, we will also work to make the code as computationally efficient as possible. For the time being, however, we advise the user to find how many data points the SCFO solver can comfortably handle in the time allotted for their particular problem, and to make that the cutoff point for the amount of data that may be fed to the solver.

5.5 The Solver Does Not Converge

If the solver has been operating for an extended number of iterations and continues to take large steps without making apparent progress or settling in a neighborhood, the most likely cause is that the quadratic bound constants M are poorly chosen – the lower ones should be pushed lower and the upper ones should be pushed higher.

6 Miscellaneous

6.1 Legal and Licensing Issues

This solver and all of its contents (i.e., those in the `.rar` archive) are free to use, free to distribute, and free to modify as you find suitable to your personal needs or application. We invite users to experiment and write their own extensions/improvements to the solver or its subroutines. Any derivative works must make reference to this original document, however, either in the documentation that accompanies them or in the preamble of the code. Neither the intellectual content nor the code of this solver may be reproduced in commercial form.

You are not required to cite this document if you use our solver in any applications the results of which are later reported orally or in writing. In the case of academic papers, we do encourage you to do so, as this provides us with an easy way of keeping track of the solver's applicability and its successes and failures.

There is no warranty that comes with this solver. We genuinely hope that it serves you well and works for your application, but cannot promise this and will not take any responsibility if it fails in any particular case, nor be held responsible for any other consequences of its implementation.

6.2 Version History

The following is a log to all of the changes that have been implemented in the solver since its initial release, and should be of interest to users of previous versions as upgrading may lead to compatability issues. All of the older versions of the solver (as well as of this guide) should be available online together with the current version – however, users are warned that older versions are no longer supported and are encouraged to use the most recent version available. We highly recommend that the user do a clean removal of the files corresponding to a previous version if a new version is downloaded so as to avoid potential mix-ups in which files MATLAB/Octave uses.

- Version 0.9 (May 22, 2013 to June 18, 2013): Initial release.
- Version 0.9.1 (June 18, 2013 to July 5, 2013):
  - Support for GNU Octave was added.
  - The way in which sufficient excitation (formerly called "minimal excitation") is implemented was changed, with the current implementation considering a metric of well-poisedness for linear regression (an expected improvement over the past metric of minimal excitation in each individual direction). Unlike in the initial release, the parameter δ_e is no longer an input but is chosen adaptively by the solver, which, based on our experience, has led to improvements in convergence speed when compared to the initial version.
- Version 0.9.2 (July 5, 2013 to August 2, 2013):
  - The log-concave approximation algorithm was replaced by a faster version that does not need to solve an optimization problem to build an approximation. Instead, a convex envelope for the noise data is constructed using the algorithm proposed in [3]. Instead of using 20 pieces to approximate the penalty function, which was, in

many cases, inefficient and sometimes led to numerical issues, only 5 pieces are used now. These are chosen by an iterative algorithm that attempts to achieve the best approximation possible with only 5 pieces (i.e., by starting with one piece, noting where the approximation error is largest, adding the next piece there, and so on).

  - The user no longer needs to provide structural assumptions to the solver, as these are now automatically chosen as linear, diagonal-quadratic, or full-quadratic depending on how many measurements are available.

  - The level of display provided by the solver can now be chosen by the user.

  - Some important changes were made to the way in which sufficient excitation is implemented. As it was noted that the SCFO was often unable to provide a good amount of excitation in problems with active constraints, the definition of $\overline{\delta}_e$ was changed to not account for the inequality constraints – instead of trying to guarantee that a ball of radius δ_e lie inside the feasible set, δ_e is first chosen to be as large as needed and numerous excitation points are tested and retained if they can be proven feasible. This allows for less conservatism and for larger excitation steps being taken in general, which aids significantly in the gradient estimation subroutines. However, since the resulting excitation steps now tend to be larger, we have decided to reduce the frequency and size of these excitations by (a) using 5 instead of 3 as the number of consecutive iterations for which the insufficient-excitation criteria should be satisfied to force excitation, and (b) choosing the desired δ_e as the step that results in a predicted signal-to-noise ratio of 0.5 (instead of 1.0, as was done earlier). We have found these choices to lead to notable improvements for several problems. Back-offs on the box constraints, which led to worse performance for problems where these were active, have been removed since they are not really needed (an orthant with sufficient excitation in all directions will always exist with respect to the box constraints).

- Version 0.9.3 (August 2, 2013 to August 18, 2013):

  - An automatic consistency check for the Lipschitz constants was added to the solver's subroutines. While we still encourage users to be careful with their choices of Lipschitz constants, this check should nevertheless be a major step forward towards easing the configuration burden of the solver and in robustifying its performance in applications. This is only implemented for the uncertain ("p") functions at the moment, as we expect the user to be capable of computing valid Lipschitz constants for the certain constraints $\mathbf{G}$.

  - The option of the solver automatically stopping adaptation of the inputs when a sufficiently optimal input set has been found has been made available. This is expected to be practical for those applications where the user has good knowledge of the minimum cost that they would like to achieve, and are willing to accept a certain amount of suboptimality upon convergence.

- Version 0.9.4 (August 18, 2013 to present):

  - The "fast" version of the solver was introduced, with the gradient estimation optimization routines replaced by faster (albeit less rigorous) least-squares regression methods. Whether the standard version or the fast version is used may be set by the user through an additional configuration parameter (Section 2.2.14).

  - An additional regularization subroutine has been added to aid with the choice of

reference point (Section 3.9).

– Some additional safeguards have been added when doing the consistency check for the Lipschitz constants, and the way in which the constants are augmented has been modified (Section 3.10).

6.3 Contact Us

We are always glad to hear from you regarding your experiences with this solver. This covers a number of points, from simple questions that were not clearly answered here, to suggestions/criticisms/(praise?), to reports of the solver's success or failure in a certain application, to a desire to seriously contribute to further development. We are open to just about anything, so please do not hesitate to write.

For all technical queries, please contact Gene Bunin at either `gene.bunin@epfl.ch` (preferred but temporary) or `eugene.bunin@gmail.com` (less preferred but more permanent). For all other queries, please contact either Dr. Grégory François (`gregory.francois @epfl.ch`) or Professor Dominique Bonvin (`dominique.bonvin@epfl.ch`).

6.4 Acknowledgements

The authors would like to thank and acknowledge:

- Ioritz Alberdi Balentziaga, for being among the first to test this solver in the response-surface/design-of-experiments context,
- Sean Costello, for providing the simulation files for the example in Section 4.1.2,
- Professor Christos Georgakis, for providing the simulation files for the example in Section 4.2.2,
- Professor Colin Jones, for initially making us conscious of the applicability of the solver to numerical problems with expensive function evaluations.

7 Bibliography

[1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2008.

[2] M. A. Brdys and P. Tatjewski. *Iterative Algorithms for Multilayer Optimizing Control*. Imperial College Press, 2005.

[3] G. A. Bunin. Extended reverse convex programming: An active-set approach to global optimization. *J. Glob. Optim. (submitted)*, 2013.

[4] G. A. Bunin, F. Fraire, G. François, and D. Bonvin. Run-to-run MPC tuning via gradient descent. In *22nd European Symposium on Computer Aided Process Engineering (London)*, pages 927–931, 2012.

[5] G. A. Bunin, G. François, and D. Bonvin. From discrete measurements to bounded gradient estimates: A look at some regularizing structures. *Ind. Eng. Chem. Res.*, 2013. doi: 10.1021/ie303309a.

[6] G. A. Bunin, G. François, and D. Bonvin. A real-time optimization framework for the iterative controller tuning problem. *Processes (submitted)*, 2013.

[7] G. A. Bunin, G. François, and D. Bonvin. Sufficient conditions for feasibility and optimality of real-time optimization schemes - III. Sufficient excitation, degradation effects, and input noise. *Comput. Chem. Eng. (in preparation)*, 2013.

[8] G. A. Bunin, G. François, and D. Bonvin. Sufficient conditions for feasibility and optimality of real-time optimization schemes - I. Theoretical foundations. *Comput. Chem. Eng. (submitted)*, 2013.

[9] G. A. Bunin, G. François, and D. Bonvin. Sufficient conditions for feasibility and optimality of real-time optimization schemes - II. Implementation issues. *Comput. Chem. Eng. (submitted)*, 2013.

[10] G. A. Bunin, F. V. Lima, C. Georgakis, and C. M. Hunt. Model predictive control and dynamic operability studies in a stirred tank: Rapid temperature cycling for crystallization. *Chem. Eng. Commun.*, 197(5):733–752, 2010.

[11] G. A. Bunin, Z. Wuillemin, G. François, A. Nakajo, L. Tsikonis, and D. Bonvin. Experimental real-time optimization of a solid oxide fuel cell stack via constraint adaptation. *Energy*, 39:54–62, 2012.

[12] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. Cambridge University Press, 2009.

[13] Wikipedia contributors. Convex function. http://en.wikipedia.org/wiki/Convex_function, March 2013.

[14] S. Engell. Feedback control for optimal process operation. *J. Process Control*, 17:203–219, 2007.

[15] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Gümüs, S. T. Harding, J. L. Klepeis, C. A. Meyer, and C. A. Schweiger. *Handbook of Test Problems in Local and Global Optimization*. Kluwer Academic Publishers, 1999.

[16] J. F. Forbes, T. E. Marlin, and J. F. MacGregor. Model adequacy requirements for optimizing plant operations. *Comput. Chem. Eng.*, 18(6):497–510, 1994.

[17] G. François, B. Srinivasan, and D. Bonvin. Use of measurements for enforcing the necessary conditions of optimality in the presence of constraints and uncertainty. *J. Process Control*, 15(6):701–712, 2005.

[18] C. Georgakis. A model-free methodology for the optimization of batch processes: Design of dynamic experiments. In *7th IFAC International Symposium on Advanced Control of Chemical Processes (ADCHEM) (Istanbul)*, pages 644–649, 2009.

[19] CVX Research Inc. *CVX: MATLAB software for disciplined convex programming, version 2.0 beta*. http://cvxr.com/cvx, September 2012.

[20] J. V. Kadam, M. Schlegel, B. Srinivasan, D. Bonvin, and W. Marquardt. Dynamic optimization in the presence of uncertainty: From off-line nominal solution to measurement-based implementation. *J. Process Control*, 17(5):389–398, 2007.

[21] N. J. Killingsworth and M. Krstić. PID tuning using extremum seeking: Online, model-free performance optimization. *Control Systems, IEEE*, 26(1):70–79, 2006.

[22] O. Lequin, E. Bosmans, and T. Triest. Iterative feedback tuning of PID parameters: Comparison with classical tuning rules. *Contr. Eng. Pract.*, 11(9):1023–1033, 2003.

[23] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook. *Response Surface Methodology*. John Wiley & Sons, 2009.

[24] R. Scattolini. Architectures for distributed and hierarchical model predictive control - a review. *J. Process Control*, 19(5):723–731, 2009.

[25] B. Srinivasan, D. Bonvin, E. Visser, and S. Palanki. Dynamic optimization of batch processes: II. Role of measurements in handling uncertainty. *Comput. Chem. Eng.*, 27(1):27–44, 2003.

[26] B. Srinivasan, S. Palanki, and D. Bonvin. Dynamic optimization of batch processes: I. Characterization of the nominal solution. *Comput. Chem. Eng.*, 27(1):1–26, 2003.

[27] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods Softw.*, 11(1-4):625–653, 1999.

[28] K. C. Toh, M. J. Todd, and R. H. Tutuncu. SDPT3 – a Matlab software package for semidefinite programming. *Optim. Methods Softw.*, 11:545–581, 1999.

[29] Y. Wang, F. Gao, and F. J. Doyle III. Survey on iterative learning control, repetitive control, and run-to-run control. *J. Process Control*, 19(10):1589–1600, 2009.

[30] T. J. Williams and R. E. Otto. A generalized chemical processing model for the investigation of computer control. *AIEE Trans.*, 79(5):458–473, 1960.