Title
Uncertainty Quantification with Experimental Data and Complex System Models

Author
Russi, Trent Michael

Publication Date
2010

Peer reviewed|Thesis/dissertation

Uncertainty Quantification with Experimental Data and Complex System Models

by

Trent Michael Russi

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering-Mechanical Engineering

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Andrew K. Packard, Co-Chair
Professor Michael Frenklach, Co-Chair
Professor Laurent El Ghaoui

Spring 2010

Uncertainty Quantification with Experimental Data and Complex System Models

by

Trent Michael Russi

Abstract

Uncertainty Quantification with Experimental Data and Complex System Models

by

Trent Michael Russi

Doctor of Philosophy in Engineering-Mechanical Engineering

University of California, Berkeley

Professor Andrew K. Packard, Co-Chair

Professor Michael Frenklach, Co-Chair

This dissertation discusses uncertainty quantification as posed in the Data Collaboration framework. Data Collaboration is a methodology for combining experimental data and system models to induce constraints on a set of uncertain system parameters. The framework is summarized, including outlines of notation and techniques. The main techniques include polynomial optimization and surrogate modeling to ascertain the consistency of all data and models as well as propagate uncertainty in the form of a model prediction.

One of the main methods of Data Collaboration is using techniques of nonconvex quadratically constrained quadratic programming to provide both lower and upper bounds on the various objectives. The Lagrangian dual of the NQCQP provides both an outer bound to the optimal objective as well as Lagrange multipliers. These multipliers act as sensitivity measures relaying the effects of changes to the parameter constraint bounds on the optimal objective. These multipliers are rewritten to provide the sensitivity to uncertainty in the response prediction with respect to uncertainty in the parameters and experimental data.

It is often of interest to find a vector of parameters that is both feasible and representative of the current community work and knowledge. This is posed as the problem of finding the minimal number of parameters that must deviate from their literature value to achieve concurrence with all experimental data constraints. This problem is heuristically solved using the ℓ^1-norm in place of the cardinality function. A lower bound on the objective is provided through an NQCQP formulation.

In order to use the NQCQP techniques, the system models need to have quadratic forms. When they do not have quadratic forms, surrogate models are fitted. Surrogate modeling can be difficult for complex models with large numbers of parameters and long simulation times because of the amount of evaluation-time required to make a good fit. New techniques are developed for searching for an active subspace of the parameters, and subsequently creating an experiment design on the active subspace

that adheres to the original parameter constraints. The active subspace can have a dimension significantly lower than the original parameter dimension thereby reducing the computational complexity of generating the surrogate model. The technique is demonstrated on several examples from combustion chemistry and biology.

Several other applications of the Data Collaboration framework are presented. They are used to demonstrate the complexity of describing a high dimensional feasible set of parameter values as constrained by experimental data. Approximating the feasible set can lead to a simple description, but the predictive capability of such a set is significantly reduced compared to the actual feasible set. This is demonstrated on an example from combustion chemistry.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to thank so many people that have helped me in various ways throughout my graduate education. From the faculty teaching my coursework to the campus staff members who helped keep all of my paperwork in order, the employees of the university helped keep me on track and focused on my goals. From my labmates to my classmates, the diverse group of people that make up my peers on campus helped me through difficult homework assignments and difficult research problems. All of these people made my day-to-day experience at UC Berkeley that much richer and more fulfilling.

I would especially like to thank my advisors, Professors Andy Packard and Michael Frenklach. Not only did they lay the groundwork for my research, but they helped direct my efforts toward the important issues. Michael taught me that there is no magical answer, and there are always trade-offs. He always says there is "no free lunch." He helped to keep me looking at the bigger picture, and helped me make sure that I was always telling a story when presenting my work. Andy was always coming up with new ideas, new insights, and new things to try. He was not afraid to get his hands dirty and help debug code. Finally, Andy tried to teach me that when things weren't working out, I had to know when to "punt."

I also want to acknowledge Ryan Feeley. Ryan was the graduate student working on this research when I started my graduate studies. He was always friendly and worked hard to help me understand the work. He wrote much of the Data Collaboration toolbox that runs a lot of our analysis. Even after he graduated, he was still willing to find time to help me dig through code and understand some of his insights.

My parents always provided so much support and encouragement. They have always believed in me and supported me. Their love and well-wishes kept me going much of the time.

And finally, thank you to Sara, who is the most patient of girlfriends. She kept a smile on my face and a fire under my butt as I pushed to finish. Sara, I'm done!

This work was supported by the NSF Information Technology Research Program, Grant No. CTS-0113985, and the NSF initiative on Collaborative Research: Cyberinfrastructure and Research Facilities, Grant No. CHE-0535542.

Chapter 1

Introduction

The American mathematician John Allen Paulos said, "Uncertainty is the only certainty there is, and knowing how to live with insecurity is the only security." The budding field of uncertainty quantification is the product of this philosophy and those like it. More and more researchers in various fields are realizing that they cannot eliminate uncertainty in their experiments and methods. The goal should be to quantify, analyze, and understand the uncertainty present in a system.

Uncertainty quantification (UQ) is a field of study pertaining to the characterization of error and uncertainty in various applications. The types of uncertainty generally are placed into either one or both of two categories. The first type is probabilistic or statistical uncertainty. In this case, uncertainties are modeled as random processes [e.g. 65, 76]. The second type is systematic uncertainty, often modeled as hard bounds on a set of possible values [e.g. 6, 84]. Some systems have both types of uncertainty. UQ attempts to build models for these uncertainties and uses experimental data to update these models.

One common task in uncertainty quantification involves using the uncertainty models of a set of system parameters to produce a measure of uncertainty on the output of a function or model [35, 57, 77]. This is known as the propagation of uncertainty. With statistical uncertainty, the Monte Carlo method is a common technique [64, 69]. With systematic uncertainty, a simple propagation using interval mathematics is common [10]. In both cases, complex systems with large numbers of parameters and long simulation times prove to be very difficult to work with. There is much recent work examining techniques for dealing with large computational complexity with regards to UQ [23, 117].

The pursuit of knowledge with regards to uncertainty is heavily dependent on data. Experimental data, while it has uncertainty of its own, can provide information about uncertainties in a system. For example, the prior knowledge on a parameter may have lots of uncertainty, modeled as either a large variance or a wide interval.

Data can help tighten this uncertainty, whether it be through statistical conditioning or by adding hard constraints. In some cases, it can correct uncertainty that was assessed incorrectly [12]. Oftentimes, experimenters are not able to directly measure the parameter of interest, but what they measure does depend on this parameter. Therefore, relating experimental data and uncertainty with parameter uncertainty also involves the propagation of uncertainty, this time in the reverse direction.

The study of the effect of data on uncertainty lies in information theory as classically described by Shannon [103]. Shannon poses information as a lack of statistical entropy, or as being the opposite of uncertainty in the prediction of an event. These ideas have since been furthered by many to include measures for the information gain specifically provided by experimentation [63] and more general notions of uncertainty [56]. This interaction between data and uncertainty depends on the underlying system models, thus modeling becomes an important part of UQ.

Propagation of errors depends on good models relating parameters to the system feature of interest, be it the observable in an experiment or some implicit system property. Modeling is one of the many ways that researchers learn about a system [58]. A model represents the detailed knowledge of a system that can be validated or invalidated with data. Good models can also provide some information by making predictions of unmeasured system properties. It is for this reason that many strive for more detailed, precise, and accurate models. Often times, however, this leads to more complexity and hence more difficult analysis.

To compensate for this increase in complexity, algebraic surrogate models are often created. Surrogate models have the benefit of increased evaluation speed and the simplicity for easier system analysis. In the case of optimization, surrogate models can provide functional forms that create optimization problems that are easier to solve [29]. Good experiment designs, especially in the presence of uncertainty, help to limit the number of evaluations of the "real" model that are required to create good surrogates [1]. The validation of models (either the original simulation models or their surrogates) also relies heavily on uncertainty quantification. Typically the models are validated if they match measured data within the measurement uncertainty [26, 78, 119].

Computing power today has greatly improved the ability to sift through large amounts of data and use it to makes increasingly precise assertions. Moreover, Moore's law suggests that computer power will continue to grow exponentially [71]. The capacity of data storage devices has also rapidly increased, even as the cost per unit capacity has decreased.[1] As scientific techniques become more sophisticated and precise, they use more of the available computing power and storage. This naturally leads to a need for more structure in the organization of scientific data [72, 85]. The scientific community is calling for not only better data organization [50, 93], but also for better collaboration in terms of providing data and not just results [19, 40, 74, 100].

This dissertation discusses some techniques for uncertainty quantification using

[1]The rate of data storage capacity increase is sometimes refered to as Kyrder's Law [116].

the information and knowledge gained from experimentation and modeling. Uncertainty on a parameter or experimentally measured value is modeled as a closed interval with no statistical interpretation. The presented method of uncertainty quantification uses constrained optimization techniques to provide bounds on propagated uncertainty. In the case when all process models are quadratic functions of the uncertain variables, outer bounds (a lower bound on the minimum and an upper bound on the maximum) are computed using techniques from control theory [22]. However, the vast majority of parameterized models are not quadratic and, as mentioned, are often complex and require long evaluation or simulation times. It is for this reason, that (piecewise) quadratic surrogate models are generated when needed. But again, with complex models this can be time-consuming and difficult, so the dissertation addresses these issues by searching for an active subspace dependence and using it to fit in smaller dimensions.

Data Collaboration is a term we use to describe our technique of combining experimental data, uncertainty, and models to constrain the correlated uncertainty in a set of parameters. The framework and method is described in Chapter 2. Chapter 3 provides a review of nonconvex quadratically constrained quadratic programming (NQCQPs). Lower bounds from NQCQP techniques provide Lagrange multipliers which provide solution sensitivities to the uncertainty in the problem. This is described in Chapter 4. A method for finding a representative vector of parameter values that matches all available experiment data and attempts to minimize the number of parameters deviated from literature values is presented in Chapter 5.

The largest new contributions are presented in Chapters 6 and 7 which focus on discovering the active subspace of a function and the subsequent experiment design on the active subspace for building a surrogate model. Chapter 8 demonstrates a few examples. Chapter 9 presents a few new applications for the Data Collaboration methodology, and Chapter 10 uses these to further explore the complexities of high dimensional systems analysis and the effects of uncertainty and approximation error therein.

Chapter 2

Data Collaboration

Experimentation and modeling are two of the most important aspects of research. Experiments help scientists understand a system's behavior, while models help predict that behavior. Furthermore, experimental data can help improve predictive models via model validation. In turn, better models lead to more understanding of system behavior and can drive the design of future experiments. Our previous work took this concept one step further by combining the experimental data and models from many different experiments on a common system [42, 43]. That methodology, called *Data Collaboration*, is summarized in this chapter.

2.1 Setup and Nomenclature

An experiment provides a measurement d of some scalar observable Y, as well as lower and upper uncertainties $l < 0$ and $u > 0$. An associated parameterized process model M predicts the measurement of Y. M is functionally dependent on an n-dimensional vector of uncertain parameters, $\mathbf{x}$. Each parameter x_i has *prior knowledge* bounds provided by domain experts.

$$\alpha_i \leq x_i \leq \beta_i \tag{2.1}$$

These parameter bounds form a n-dimensional orthotope (a "hyperrectangle") denoted $\mathcal{H}$. Combining all of the models, data, and uncertainty for an experiment forms *a dataset unit*. The collection of many experiments along with the prior knowledge $\mathcal{H}$, forms a *dataset* $\mathcal{D}$. While the parameter values are constrained to the prior bounds hyperrectangle, the dataset unit for each experiment, indexed by e, further constrains the parameter values, via

$$l_e \leq M_e(\mathbf{x}) - d_e \leq u_e. \tag{2.2}$$

The set of valid parameters, those satisfying all constraints, is known as the *feasible set* $\mathcal{F}$.

The dataset and resulting feasible set represent the collective (collaborative) knowledge about the system. We use this knowledge as the starting point for asking questions about the system. Are the data and models from the various experiments mathematically compatible? What is the range that a predictive model can take over the feasible parameters? These questions are addressed in Sections 2.2 and 2.3, respectively, where they are first posed as optimization problems.

2.2 Dataset Consistency

If the feasible set $\mathcal{F}$ for a given dataset $\mathcal{D}$ is empty, then there is no single vector $\mathbf{x}$ in the prior knowledge bounds $\mathcal{H}$ that satisfies all experiment constraints (equation (2.2)). In this situation, we say the dataset is inconsistent. This could be due to errors in any part of the dataset, such as the experimental measurement, experiment uncertainties, prior bounds, or models. We can ascertain a dataset's consistency by solving the following feasibility problem:

$$\textit{Is the set } \mathcal{F} := \{\mathbf{x} \in \mathcal{H} \ : \ l_e \leq M_e(\mathbf{x}) - d_e \leq u_e, \ e = 1, \ldots, m\} \textit{ nonempty?} \tag{2.3}$$

Data Collaboration techniques take this concept one step further. Previous work introduced the *consistency measure* of a dataset $\mathcal{D}$, notated $\mathrm{C}_{\mathcal{D}}$, which is defined as the maximum amount that all experiment uncertainties can be reduced such that the dataset is consistent. Originally, the definition involved the absolute uncertainty reduction [32, 33], and was later redefined as the relative uncertainty decrease [34], which we present here.

$$\begin{aligned} \mathrm{C}_{\mathcal{D}} := & \max_{\mathbf{x},\gamma} \gamma \\ & \text{s.t.} \ \begin{cases} \mathbf{x} \in \mathcal{H} \\ l_e(1-\gamma) \leq M_e(\mathbf{x}) - d_e \leq u_e(1-\gamma), \quad e = 1, \ldots, m. \end{cases} \end{aligned} \tag{2.4}$$

Notice that when $\mathrm{C}_{\mathcal{D}}$ is positive ($\gamma > 0$) the experiment bounds can be tightened and a feasible point is still found. This implies there was a feasible point to begin with. Thus $\mathrm{C}_{\mathcal{D}} \geq 0$ implies consistency. Likewise, a negative consistency measure implies the constraints have to be loosened in order to find a feasible point, and hence the dataset is inconsistent.

The consistency measure provides a quantitative measure for the "degree" of consistency. The larger $\mathrm{C}_{\mathcal{D}}$ is, the larger the agreement between the information provided by each experiment (i.e. the larger the intersection of each experiment's valid parameter set). Along with the associated Lagrange multipliers (see Chapter 4), this measure can be helpful in determining which experiments add the most information to the dataset, which experiment causes an inconsistency, and discrimination of models [see 32, 33, 47].

2.3 Response Prediction

Response prediction is the prediction of the range of values an observable's model can output given assumptions on the possible values of its inputs. Often the input parameters are assumed to vary within an interval or are treated as a random variable with a given distribution. Furthermore, input parameters are often assumed to be independent. However, Data Collaboration techniques show that by constraining the parameter space using experimental data and models, we can tighten our assumptions on the parameter space. Specifically, we only consider parameter values that are in the feasible set $\mathcal{F}$.

Take a new observable's model $M_0 : \mathbb{R}^n \to \mathbb{R}$. The model maps a vector in the parameter space and returns a scalar attribute (observable). We wish to find both the minimum and maximum values that M_0 outputs over inputs from the feasible set $\mathcal{F}$. This pair of optimizations is the response prediction problem.

$$L_0 := \min_{\mathbf{x}} M_0(x) \quad \text{s.t.} \begin{cases} \mathbf{x} \in \mathcal{H} \\ l_e \leq M_e(\mathbf{x}) - d_e \leq u_e, \quad e = 1, \ldots, m \end{cases} \tag{2.5}$$

$$R_0 := \max_{\mathbf{x}} M_0(x) \quad \text{s.t.} \begin{cases} \mathbf{x} \in \mathcal{H} \\ l_e \leq M_e(\mathbf{x}) - d_e \leq u_e, \quad e = 1, \ldots, m \end{cases} \tag{2.6}$$

The resulting prediction of M_0 is the interval $[L_0, R_0]$.

Information added to the dataset can potentially improve the prediction. If more experiments are added, or uncertainty bounds are tightened, then the feasible set may shrink, resulting in a possible tightening of the prediction interval. It is important to note that dataset consistency is an important requirement for a response prediction. If the feasible set is empty, i.e. the dataset is inconsistent, then many optimization software packages will report the nonsensical interval prediction $[\infty, -\infty]$.

2.4 Techniques

For each of the optimal objectives $\mathrm{C}_{\mathcal{D}}$, L_0, and R_0, we are able to compute both upper and lower bounds. The lower bound on a minimumization and the upper bound on a maximumization are referred to as *outer bounds*. To compute these, our main techniques include surrogate model approximations, polynomial optimization, semi-definite programming, and branch and bound algorithms. The outer bounds are notated as $\overline{\mathrm{C}}_{\mathcal{D}}$, $\underline{L}_0$, and $\overline{R}_0$. In some situations, outer bounds are required calculations. For example, to prove inconsistency the outer (upper) bound on the consistency measure $\mathrm{C}_{\mathcal{D}}$ must be negative. The computation of these outer bounds discussed in Sections 2.4.1, 2.4.2, and 2.4.3.

Likewise, the upper bound on a minimization and the lower bound on a maximumization are called *inner bounds*. These are computed with interior-point methods. Specifically, we use the `fmincon` algorithm which is part of the MATLAB optimization toolbox. We notate the inner bounds with appropriate overlines and underlines as $\underline{\mathrm{C}}_{\mathcal{D}}$, $\overline{L}_0$, and $\underline{R}_0$. Again, some situations require inner bounds. For example, to prove consistency of a dataset, the inner (lower) bound on the consistency measure $\mathrm{C}_{\mathcal{D}}$ must be positive.

2.4.1 NQCQP Formulation

In the event that all models involved in a dataset are quadratic, the consistency measure (equation (2.4)) is a nonconvex quadratically constrained quadratic program (NQCQP). Furthermore, if a predictive model of interest M_0 is also quadratic, then the response prediction (equations (2.5) and (2.6)) is formed by two NQCQPs. These problems can be written as the primal NQCQP problem

$$\begin{aligned} p^\star := \min_{\mathbf{x}} \; & \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Z}_0 \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \\ \text{s.t.} \; & \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \leq 0, \quad i = 1, \ldots, 2n + 2m \end{aligned} \tag{2.7}$$

where $p^\star$ is the optimal primal solution, each $\mathbf{Z}_i$ (including $i = 0$) is a $(n+1) \times (n+1)$ matrix. Each of the $2n$ parameter constraints and $2m$ experiment constraints can be written as one of the $\left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \leq 0$ constraints. Maximization problems can also be written as a minimization, by changing the sign of the objective function.

There are existing methods for computing both upper and lower bounds on the optimal value $p^\star$ [38, 102]. Outer bounds are computed by relaxing the original problem to form a semidefinite program (SDP). Both inner and outer bound solutions for NQCQPs will be discussed in further detail in Chapter 3. To decrease the gap between inner and outer bounds we use branch and bound techniques, which are described briefly in Section 2.4.3.

Techniques have also been developed for rational quadratic forms (the ratio of two quadratic functions) [34]; however, this document focuses on optimization with quadratic models.

2.4.2 Surrogate Model Fitting

When the models are not quadratic, one can fit quadratic functions to the models in order to utilize NQCQP techniques. These types of "replacement" functions are called *surrogate models* for each model M_e and are denoted as S_e. This technique is also known as solution mapping and as the response surface methodology [44, 75, 102]. The surrogate for model e is created by sampling the prior knowledge set of parameters $\mathcal{H}$, evaluating M_e at these points, and fitting a quadratic model to

this data. Depending on the behavior of M_e, this may require a large number of evaluations for the fit to converge. However, even if the fit has converged, it may still be an inaccurate fit if the underlying model is significantly non-quadratic. One way to improve the fit in this instance is to use a branch and bound technique (see §2.4.3).

The use of many function evaluations can be especially problematic if each one takes a significant amount of computation time to produce. One way to cope with this problem is to fit an intermediate surrogate model. This intermediate model would not have to be quadratic, but could take any form that fits the data well and is inexpensive to evaluate. Ideally, this intermediate model would require less function evaluations to generate than a quadratic model, and would have much less fitting error. A quadratic surrogate model is then made to approximate the intermediate model. Since the intermediate model is quick to evaluate, the number of function evaluations needed to make the quadratic fit is less of an issue. See [34] for details.

The dimensionality of the problem quickly becomes an issue when making surrogate models. To sufficiently sample a high-dimensional space, the number of points should increase exponentially. As the dimension of the problem grows, the needed sample size rapidly becomes unmanageable. For example, with only two grid points per dimension, a 30-parameter space would have over a billion samples. As a result, when creating surrogate models we are forced to under sample. To attempt to decrease the number of samples needed, we developed a method for searching for an underlying lower-dimensional subspace dependence in a function (see Chapter 6).

2.4.3 Branch and Bound Algorithms

Branch and bound algorithms are used both to reduce the gap between inner and outer bounds [15, 16] as well as to improve the fitting error of the surrogate models. This class of algorithms has long been used to compute quantities of interest over uncertain parameters [e.g. 9]. The algorithms generally consist of two main steps that are iterated. First, in the branching step, the domain of the problem is subdivided to create smaller subproblems with a presumably higher degree of accuracy than the original problem. Second, the objective is bounded above and below. This process is then iterated to converge the bounds.

To improve surrogate fitting, the domain is divided until the model is sufficiently fitted with a quadratic over each subdomain. This procedure produces a piecewise quadratic surrogate model. Inner and outer bounds are then found for each subdomain. Our experience shows that the outer bounds tend to be less accurate than the inner bounds, even with accurrate and precise surrogate fits. To improve the outer bound approximations we can continue to subdivide the domain, without necessarily refitting the surrogate [32, 34]. Heuristics for a good choice of a subdivision have been developed [34] and can significantly improve the outer bounds.

2.5 Summary of Notation

For reference, Table 2.1 provides a list of much of the notation introduced in this chapter that will be used throughout the rest of this document. Very nearly the same notation appears in past publications, with [34] being the most similar.

Table 2.1: List of notation introduced in this chapter and used throughout this document.

Symbol	Datatype	Description
e	scalar	Index for experiments
d_e	scalar	Observable value for experiment e
l_e	scalar	Lower uncertainty bound for experiment e
u_e	scalar	Upper uncertainty bound for experiment e
M_e	function	Parameterized model outputting the scalar observable associated with experiment e
$\mathbf{x}$	$n \times 1$ vector	Vector of parameters
α_i	scalar	Lower bound of the value of the i^{th} parameter
β_i	scalar	Upper bound of the value of the i^{th} parameter
$\mathcal{H}$	set	Set of prior knowledge parameters: $\mathcal{H} := \{x \,\vert\, \alpha_i \leq x_i \leq \beta_i, \quad i = 1, \ldots, n\}$
$\mathcal{D}$	dataset	Collection of models, observations with uncertainty, and the prior knowledge $\mathcal{H}$
$\mathcal{F}$	set	Feasible set of valid parameters as defined by parameter and experiment bounds: $\mathcal{F} := \{x \in \mathcal{H} \,\vert\, l_e \leq M_e(\mathbf{x}) - d_e \leq u_e, \quad e = 1, \ldots, m\}$
$\mathrm{C}_{\mathcal{D}}$	scalar	Consistency measure for dataset D. A positive value implies consistency and negative value implies inconsistency. See equation (2.4).
$\underline{\mathrm{C}}_{\mathcal{D}}$	scalar	Lower bound on $\mathrm{C}_{\mathcal{D}}$ from interior point methods.
$\overline{\mathrm{C}}_{\mathcal{D}}$	scalar	Upper bound on $\mathrm{C}_{\mathcal{D}}$ from the S-procedure.
L_0	scalar	Left bound of the model M_0 as determined from the response prediction. See equation (2.5).
R_0	scalar	Right bound of the model M_0 as determined from the response prediction. See equation (2.6).
$\underline{L}_0$	scalar	Lower (outer) bound on the left prediction bound L_0.
$\overline{L}_0$	scalar	Upper (inner) bound on the left prediction bound L_0.
$\underline{R}_0$	scalar	Lower (inner) bound on the right prediction bound R_0.
$\overline{R}_0$	scalar	Upper (outer) bound on the right prediction bound R_0.

2.6 Example: GRI-Mech 3.0 Dataset

The GRI-Mech 3.0 dataset is a collection of process models and experimentally measured data related to the combustion of methane [106]. The dataset has 102 uncertain parameters that are mostly constants from reaction rate equations. Prior information bounds on the parameters are provided from experimentation and the combustion literature. Each parameter has been transformed such that it is constrained to the bounds $[-1,\ 1]$. The dataset also contains models and experimental observation data for 77 observables, referred to as "targets." The working models used by the GRI-Mech team are already fitted with quadratic response surfaces.

The dataset, as presented on the GRI-Mech website [106], does not include experimental uncertainty bounds. In previous work, the reasonable but otherwise arbitrary uncertainty of 0.1 was applied to all experiment observations [32, 33, 95], and that is what we use for the following examples. However, recently, Xiaoqing You, a domain expert, has decided upon uncertainties for each target based on the type of experiment and her expertise [120, 121]. We will explore examples with these uncertainties in later Sections (e.g. see §4.5).

We use the GRI-Mech 3.0 dataset for many examples since it is often relatively quick to compute the consistency measure and response prediction using this dataset because the models are already quadratic and therefore time is not spent making surrogate models. Also, the dataset has 102 parameters and interesting phenomenon are observed in large dimensions such as this (see Chapter 10).

Using the techniques outlined in Section 2.4, lower and upper bounds were computed for the GRI-Mech 3.0 dataset as 0.278 and 0.403. The bounds restrict $\mathrm{C}_{\mathcal{D}}$ to be positive, and hence the dataset is consistent. The bounds imply that the maximum amount each experiment uncertainty can be reduced by and still maintain feasibility is at least 27.8% and no more 40.3%.

To demonstrate the response prediction, we examine GRI-Mech 3.0 target StF8, the laminar flame speed in a stoichiometric atmospheric ethane-air mixture [113]. The results from predicting this target using the rest of the GRI-Mech 3.0 dataset constraints are shown in Table 2.2, which also includes the results after one branch and bound algorithm iteration is used to improve the outer bounds.

Table 2.2: Prediction of GRI-Mech 3.0 target StF8 using the rest of the GRI-Mech 3.0 dataset. Includes initial prediction, and results after one branch and bound algorithm iteration.

	$\underline{L}_0$	$\overline{L}_0$	$\underline{R}_0$	$\overline{R}_0$
Initial prediction	1.538	1.543	1.753	1.767
One B&B iteration	1.541	1.543	1.753	1.765

Chapter 3

Nonconvex Quadratically Constrained Quadratic Programming

Constrained optimization is the search for maximum or minimum values of an objective function subject to one or more constraints on its input. The difficulty in solving such problems relies on the functions that make up the objective and constraint functions. Typically, optimization involving convex functions and constraints are considered efficient to solve [14, 91]. In this chapter, we consider a class of constrained optimization problems involving nonconvex quadratic functions. Nonconvex quadratically constrained quadratic programs (NQCQPs) are not considered to be efficient to solve in general; however, they can be bounded using interior-point methods for local solutions and a convex relaxation using semidefinite programming. This chapter contains a quick summary of NQCQPs and some solution techniques. Most of the work presented in this chapter comes from the work of others [14, 101, 102, 108, 114]. Specifically, much of the notation and derivation comes from notes by Pete Seiler [101]; however, we present the handling of equality constraints directly.

Given quadratic surrogate models, each of the basic Data Collaboration problems, such as the consistency measure and response prediction, can be formulated as a nonconvex quadratically constrained quadratic problem. The basic form of this problem

is written as

$$p^\star := \min_{\mathbf{x}} \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Z}_0 \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \quad \text{s.t.} \begin{cases} \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \leq 0, \ i = 1, \ldots, N, \\ \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathfrak{Z}_j \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] = 0, \ j = 1, \ldots, M. \end{cases} \tag{3.1}$$

Each $\mathbf{Z}_i$ and $\mathfrak{Z}_j$ is a $(n+1) \times (n+1)$ matrix of quadratic coefficients. The equality constraints can be handled as inequality constraints by requiring the quadratic form to be both greater and less than 0. However, we will use equality constraints directly in the bounds for this problem. Note that there are no equality constraints used for the consistency measure and response prediction problems, but we will present a problem in Chapter 5 that does use them.

In the Data Collaboration framework, after the response models have been fit by quadratic functions, the feasible set is described by nonconvex quadratic constraints. There are two quadratic forms, $\mathbf{Z}_i$'s, for each experiment and each parameter constraint; one for the lower bound constraint, and one for the upper bound constraint. Several of the data collaboration problems involve maximizations, which requires simply changing the sign of the objective, minimizing, then changing the sign again. Since the problems can all be cast as minimizations, we will only examine the minimization NQCQP in this chapter.

The NQCQP formulation is NP-hard [83, 96]. This means that all problems in the class NP can be rewritten as an NQCQP in polynomial time[1] (i.e. a time that grows as a polynomial in the problem size) [105]. The class NP is the class of problems such that a solution can be verified in polynomial time [21]. NP-hard problems are, as their name implies, difficult. Garey and Johnson famously wrote that NP-hard means that "I can't find an efficient algorithm, but neither can all these famous people [46]." However, there are many tools and heuristics to efficiently solve for bounds on the optimal value of the NQCQP. These techniques are outlined in the rest of the chapter.

3.1 Lower Bound on the NQCQP

By using the Lagrangian dual and constraint relaxations we can construct optimization problems whose solutions provide lower bounds to the NQCQP. The next few sections will derive two such lower bounds using the S-procedure and a rank relaxation, and then demonstrate their relationship as dual problems. The derivations are based on those presented in [14, 91, 101].

[1] There is a simple polynomial-time reduction from the Satisfiability problem (which is NP-complete) to quadratically constrained quadratic programming.

3.1.1 Lower bound using the S-procedure

The NQCQP is lower bounded using the S-procedure which formulates the problem as a semi-definite program (SDP) [14, 22].

$$\underline{p}_{\mathrm{s}}^{\star} := \max_{\boldsymbol{\lambda},\boldsymbol{\nu},\gamma} \gamma \quad \text{s.t.} \quad \begin{cases} \mathbf{Z}_0 - \left[\begin{smallmatrix} \gamma & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + \sum_{i=1}^{N} \lambda_i \mathbf{Z}_i + \sum_{j=1}^{M} \nu_j \mathfrak{Z}_j \succeq 0 \\ \boldsymbol{\lambda} \geq 0 \end{cases} \tag{3.2}$$

The "$\succeq 0$" constraint requires that the matrix sum be positive semidefinite, that is, it is required to have only nonnegative eigenvalues. The Lagrange multipliers λ_i and ν_j provide important sensitivity information. Sensitivities will be discussed in further depth in Chapter 4.

The S-procedure formulation is in P, the class of problems whose solving time grows polynomially with the problem size [114]. This is a significant improvement over the original NQCQP formulation which is an NP-hard problem.

The derivation of the S-procedure involves formulating the Lagrangian dual problem and converting the quadratic constraints to linear matrix inequalities (semidefinite constraints). The dual problem formulation essentially involves changing the objective function to include a weighted sum of the constraints [14]. Starting with the nonconvex quadratically constrained quadratic problem in equation (3.1), the lower bound problem in equation (3.2) is derived as in [14, 101].

$$\begin{aligned}
p^{\star} &= \min_{\mathbf{x}} \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_0 \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \text{ s.t. } \begin{cases} \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \leq 0, & i = 1, \ldots, N \\ \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathfrak{Z}_j \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] = 0, & j = 1, \ldots, M \end{cases} \\
&\overset{(a)}{=} \min_{\mathbf{x}} \max_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}} \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_0 \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] + \sum_{i=1}^{N} \lambda_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] + \sum_{j=1}^{M} \nu_j \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathfrak{Z}_j \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \\
&\overset{(b)}{\geq} \max_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}} \min_{\mathbf{x}} \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_0 \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] + \sum_{i=1}^{N} \lambda_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] + \sum_{j=1}^{M} \nu_j \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathfrak{Z}_j \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \\
&\overset{(c)}{=} \max_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}, \gamma} \gamma \text{ s.t. } \min_{\mathbf{x}} \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_0 \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] + \sum_{i=1}^{N} \lambda_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] + \sum_{j=1}^{M} \nu_j \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathfrak{Z}_j \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \geq \gamma \\
&\overset{(d)}{=} \max_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}, \gamma} \gamma \text{ s.t. } \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \left(\mathbf{Z}_0 - \left[\begin{smallmatrix} \gamma & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + \sum_{i=1}^{N} \lambda_i \mathbf{Z}_i + \sum_{j=1}^{M} \nu_j \mathfrak{Z}_j \right) \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \geq 0 \ \forall \mathbf{x} \\
&\overset{(e)}{=} \max_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}, \gamma} \gamma \text{ s.t. } \mathbf{Z}_0 - \left[\begin{smallmatrix} \gamma & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + \sum_{i=1}^{N} \lambda_i \mathbf{Z}_i + \sum_{j=1}^{M} \nu_j \mathfrak{Z}_j \succeq 0 \\
&= \underline{p}_{\mathrm{s}}^{\star}
\end{aligned} \tag{3.3}$$

Equality (a) is minimizing the Lagrangian function, (b) is the relaxation due to switching the order of the minimization and maximization, (c) simply adds the slack variable

γ, (d) removes the minimization by noting that the inequality holds for all $\mathbf{x}$ if and only if it holds for the minimum, and (e) is a simple "if and only if" property shown in [101]. The derivation in equation (3.3) is known as the S-procedure. The solution $\underline{p}^\star_{\mathrm{s}}$ is a lower bound on the optimal primal solution $p^\star$. The final equality, (e), yields a semi-definite program, which is a convex optimization in its variables γ, $\boldsymbol{\lambda}$, and $\boldsymbol{\nu}$.

Since the S-procedure is a maximization problem, any feasible solution will be a lower bound on the optimum. Therefore, any feasible solution provided by a numerical solver (such as SeDuMi) will be a guaranteed lower bound on the optimal NQCQP minimization problem.

3.1.2 Lower bound using rank relaxation

Another way to achieve lower bounds on the NQCQP minimization is to relax the constraints. One choice is a formulation using a rank relaxation.

$$\begin{aligned} \underline{p}^\star_{\mathrm{r}} := \min_{\mathbf{Q}\succeq 0, Q_{11}=1} & \operatorname{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] \\ \text{s.t.} & \begin{cases} \operatorname{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] \le 0, & i=1,\ldots,N, \\ \operatorname{Tr}\left[\mathfrak{Z}_j\mathbf{Q}\right] = 0, & j=1,\ldots,M. \end{cases} \end{aligned} \tag{3.4}$$

Here, $\mathbf{Q}$ is a $(n+1)\times(n+1)$ matrix constrained to be positive semidefinite with a 1 in (1,1) entry, and Tr is the matrix trace function. The optimal variable $\mathbf{Q}^\star$ can be used for branching heuristics in branch and bound algorithms [34] as well as an interior point method seeding for an upper bound on the NQCQP (see §3.3).

The formulation of the rank relaxation is a matter of simply rewriting the objective and constraints before relaxing one of the constraints.

$$\begin{aligned} p^\star &= \min_{\mathbf{x}} \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_0 \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \text{ s.t. } \begin{cases} \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \le 0, & i=1,\ldots,N \\ \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}} \mathfrak{Z}_j \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] = 0, & j=1,\ldots,M \end{cases} \\ &\overset{(a)}{=} \min_{\mathbf{x}} \operatorname{Tr}\left[\left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_0 \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]\right] \text{ s.t. } \begin{cases} \operatorname{Tr}\left[\left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]\right] \le 0, & i=1,\ldots,N \\ \operatorname{Tr}\left[\left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}} \mathfrak{Z}_j \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]\right] = 0, & j=1,\ldots,M \end{cases} \\ &\overset{(b)}{=} \min_{\mathbf{x}} \operatorname{Tr}\left[\mathbf{Z}_0 \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}}\right] \text{ s.t. } \begin{cases} \operatorname{Tr}\left[\mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}}\right] \le 0, & i=1,\ldots,N \\ \operatorname{Tr}\left[\mathfrak{Z}_j \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}}\right] = 0, & j=1,\ldots,M \end{cases} \\ &\overset{(c)}{=} \min_{\mathbf{Q}\succeq 0, Q_{11}=1} \operatorname{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] \text{ s.t. } \begin{cases} \operatorname{rank}(\mathbf{Q}) = 1 \\ \operatorname{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] \le 0, & i=1,\ldots,N \\ \operatorname{Tr}\left[\mathfrak{Z}_j\mathbf{Q}\right] = 0, & j=1,\ldots,M \end{cases} \\ &\overset{(d)}{\ge} \min_{\mathbf{Q}\succeq 0, Q_{11}=1} \operatorname{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] \text{ s.t. } \begin{cases} \operatorname{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] \le 0, & i=1,\ldots,N \\ \operatorname{Tr}\left[\mathfrak{Z}_j\mathbf{Q}\right] = 0, & j=1,\ldots,M \end{cases} \\ &= \underline{p}^\star_{\mathrm{r}}. \end{aligned} \tag{3.5}$$

The quadratic quantities in the original problem evaluate to scalars, therefore (a) follows from the fact that the trace function does not affect scalars. Equality (b) comes from the fact that the trace is invariant under cycle permutations of the products within. Replacing the outer product of the $\mathbf{x}$ vector with the matrix $\mathbf{Q}$ yields (c). Finally, by relaxing the rank constraint (i.e. removing it) we get a lower bound problem as shown in relation (d). This formulation is also an SDP, although it is not in the standard form.

3.1.3 Duality of Lower Bound Formulations

The S-procedure formulation is also the Lagrangian dual of the rank relaxation problem. This is shown with the following set of relations.

$$
\begin{aligned}
\underline{p}^{\star}_{\mathrm{r}} &= \min_{\mathbf{Q}\succeq 0} \mathrm{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] \text{ s.t. } \begin{cases} Q_{11}=1 \\ \mathrm{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right]\le 0, & ,i=1,\ldots,N \\ \mathrm{Tr}\left[\mathfrak{Z}_i\mathbf{Q}\right]=0, & ,j=1,\ldots,M \end{cases} \\
&\overset{(a)}{=} \min_{\mathbf{Q}\succeq 0}\max_{\boldsymbol{\lambda}\ge 0,\boldsymbol{\nu},\gamma} \mathrm{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] + (1-Q_{11})\gamma + \sum_{i=1}^{N}\lambda_i \mathrm{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] + \sum_{j=1}^{M}\nu_j \mathrm{Tr}\left[\mathfrak{Z}_j\mathbf{Q}\right] \\
&\overset{(b)}{\ge} \max_{\boldsymbol{\lambda}\ge 0,\boldsymbol{\nu},\gamma}\min_{\mathbf{Q}\succeq 0} \mathrm{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] + (1-Q_{11})\gamma + \sum_{i=1}^{N}\lambda_i \mathrm{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] + \sum_{j=1}^{M}\nu_j \mathrm{Tr}\left[\mathfrak{Z}_j\mathbf{Q}\right] \\
&\overset{(c)}{=} \max_{\boldsymbol{\lambda}\ge 0,\boldsymbol{\nu},\gamma}\min_{\mathbf{Q}\succeq 0} \gamma + \mathrm{Tr}\left[\left(\mathbf{Z}_0 - \left[\begin{smallmatrix}\gamma & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n\end{smallmatrix}\right] + \sum_{i=1}^{N}\lambda_i\mathbf{Z}_i + \sum_{j=1}^{M}\nu_j\mathfrak{Z}_j\right)\mathbf{Q}\right] \\
&\overset{(d)}{=} \max_{\boldsymbol{\lambda}\ge 0,\boldsymbol{\nu},\gamma} \gamma \text{ s.t. } \mathbf{Z}_0 - \left[\begin{smallmatrix}\gamma & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n\end{smallmatrix}\right] + \sum_{i=1}^{N}\lambda_i\mathbf{Z}_i + \sum_{j=1}^{M}\nu_j\mathfrak{Z}_j \succeq 0 \\
&= \underline{p}^{\star}_{\mathrm{s}},
\end{aligned}
\tag{3.6}
$$

where (a) is minimizing the Lagrangian function, (b) is a relaxation from switching the maximization and minimization operators, (c) utilizes the linearity of the trace function, and (d) is changing the Lagrangian to a semidefinite constraint. If either the S-procedure problem or the rank relaxation problem is bounded and strictly feasible, then strong duality holds, i.e. $\underline{p}^{\star}_{\mathrm{s}} = \underline{p}^{\star}_{\mathrm{r}}$ [91]. In fact, even weaker conditions exist for strong duality that we can assume hold for most cases [108].

The duality of the S-procedure and rank relaxation problems allows them to be solved with a single call to the freely available MATLAB package SeDuMi [109]. Our experience has been that, given quadratic models, the outer bound solutions take much less computation time as compared to the inner bound computation.

3.2 Stochastic Interpretation

The rank relaxation formulation (equation (3.4)) can be written with a stochastic interpretation. To see this, first partition the optimization variable $\mathbf{Q}$ as,

$$\mathbf{Q} = \begin{bmatrix} 1 & \overline{\mathbf{x}}^{\mathsf{T}} \\ \overline{\mathbf{x}} & \mathbf{\Sigma} \end{bmatrix} \tag{3.7}$$

where $\overline{\mathbf{x}} \in \mathbb{R}^n$ and $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$.

Let $\mathbf{X}$ be a random vector with $\mathrm{E}\left[\mathbf{X}\right] = \overline{\mathbf{x}}$ and $\mathrm{E}\left[\mathbf{X}^{\mathsf{T}}\mathbf{X}\right] = \mathbf{\Sigma}$. Using the linearity of the trace and expectation functions, the trace constraints (and objective) in the rank relaxation problems can be rewritten using the random vector $\mathbf{X}$.

$$\begin{aligned} \mathrm{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] &= \mathrm{Tr}\left[\mathbf{Z}_i \left[\begin{smallmatrix} 1 & \overline{\mathbf{x}}^{\mathsf{T}} \\ \overline{\mathbf{x}} & \mathbf{\Sigma} \end{smallmatrix}\right]\right] \\ &= \mathrm{E}\left[\mathrm{Tr}\left[\mathbf{Z}_i \left[\begin{smallmatrix} 1 & \mathbf{X}^{\mathsf{T}} \\ \mathbf{X} & \mathbf{X}\mathbf{X}^{\mathsf{T}} \end{smallmatrix}\right]\right]\right] \\ &= \mathrm{E}\left[\mathrm{Tr}\left[\left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]\right]\right] \\ &= \mathrm{E}\left[\left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]\right] \end{aligned} \tag{3.8}$$

This equality gives a direct interpretation of the rank relaxation problem using random variables.

$$\begin{aligned} \underline{p}^{\star}_{\mathrm{r}} = \min_{\mathbf{X},\overline{\mathbf{x}},\mathbf{\Sigma}} \ & \mathrm{E}\left[\left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_0 \left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]\right] \\ \text{s.t.} \ & \begin{cases} \mathrm{E}\left[\left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]\right] \leq 0, \ i = 1, \ldots, N \\ \mathrm{E}\left[\left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]^{\mathsf{T}} \mathfrak{Z}_j \left[\begin{smallmatrix} 1 \\ \mathbf{X} \end{smallmatrix}\right]\right] \leq 0, \ j = 1, \ldots, M \\ \mathbf{X} \text{ is a random vector with } \mathrm{E}\left[\mathbf{X}\right] = \overline{\mathbf{x}} \ \& \ \mathrm{E}\left[\mathbf{X}\mathbf{X}^{\mathsf{T}}\right] = \mathbf{\Sigma} \end{cases} \end{aligned} \tag{3.9}$$

Here the $\mathbf{Q} \succeq 0$ constraint from the rank relaxation problem is still satisfied. The covariance matrix associated with a random vector is always positive semidefinite. The covariance is $\mathrm{E}\left[(\mathbf{X} - \overline{\mathbf{x}})(\mathbf{X} - \overline{\mathbf{x}})^{\mathsf{T}}\right] = \Sigma - \overline{\mathbf{x}}\overline{\mathbf{x}}^{\mathsf{T}}$. Using the Schur complement, $\mathbf{Q} \succeq 0$ is equivalent to $\mathbf{\Sigma} - \overline{\mathbf{x}}\overline{\mathbf{x}}^{\mathsf{T}} \succeq 0$.

The optimal $\mathbf{Q}^{\star}$ matrix from the rank relaxation problem therefore provides the mean and covariance for the optimal random vector to solve equation (3.9). These first and second order statistics will be used to compute the upper bound on the NQCQP in Section 3.3.

3.3 Upper Bound on the NQCQP

The NQCQP cost function evaluated at any feasible point (i.e. points satisfying all constraints in equation (3.1)) will be greater than or equal to the optimal value.

Therefore any feasible point provides an upper bound on the optimal value $p^\star$. Readily available computer solvers are used to tighten this upper bound down using interior point methods [102].

The previous section demonstrated an interpretation of the rank relaxation optimal variable as the first and second order statistics of an optimal random vector. Sampling from a distribution with these statistics will not necessarily provide a feasible point, but can provide a good starting (seed) point for the interior-point solvers used to find an upper bound solution.

3.4 NQCQP Formulation of Consistency and Response Prediction Problems

As briefly described in Section 2.4, Data Collaboration techniques call for fitting experiment models with quadratic surrogates. Branch and bound algorithms improve these fits, by subdividing the prior bound hyperrectangle into sub-hyperrectangles, and fitting each experiment model over each subdomain. Once reasonable fits are made, solving for the consistency measure $\mathrm{C}_{\mathcal{D}}$ and response prediction bounds L_0 and R_0 are NQCQPs on each subdomain.

We now focus on the specific formulations for a given subdomain. Let $\mathbf{S}_e$ denote the surrogate coefficient matrix for the experiment e, and $l_{e,\text{fit}}$ and $u_{e,\text{fit}}$ are the associated absolute fitting errors such that the inequality

$$l_{e,\text{fit}} \leq \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{S}_e \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] - M_e(\mathbf{x}) \leq u_{e,\text{fit}} \tag{3.10}$$

holds for all $\mathbf{x}$ in the relevant subdomain. The experiment bounds can then be rewritten as

$$l_e + l_{e,\text{fit}} \leq \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{S}_e \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] - d_e \leq u_e + u_{e,\text{fit}}. \tag{3.11}$$

For a response prediction, the objective coefficient matrix $\mathbf{Z}_0^{\text{prediction}}$ simply corresponds to the coefficients of the model observable to be predicted, $\mathbf{S}_0$. The sign of this matrix is of course flipped for the right prediction bound (maximization) problem. Each of the experiments have two inequalities, one for the left bound, and one for the right. The associated NQCQP constraint matrices are

$$\mathbf{Z}_{e,l}^{\text{prediction}} = \begin{bmatrix} l_e + l_{e,\text{fit}} + d_e & \mathbf{0}_{1\times n} \\ \mathbf{0}_{n\times 1} & \mathbf{0}_{n\times n} \end{bmatrix} - \mathbf{S}_e, \tag{3.12}$$

$$\mathbf{Z}_{e,u}^{\text{prediction}} = \mathbf{S}_e - \begin{bmatrix} u_e + u_{e,\text{fit}} + d_e & \mathbf{0}_{1\times n} \\ \mathbf{0}_{n\times 1} & \mathbf{0}_{n\times n} \end{bmatrix}. \tag{3.13}$$

For the consistency measure, there is an extra optimization variable γ. Therefore, the quadratic coefficient matrices will be $(n+2)\times(n+2)$ in size, and the quadratic form will be

$$\begin{bmatrix} 1 \\ \mathbf{x} \\ \gamma \end{bmatrix}^{\mathsf{T}} \mathbf{Z}_i \begin{bmatrix} 1 \\ \mathbf{x} \\ \gamma \end{bmatrix}. \tag{3.14}$$

The maximal γ is the consistency measure, so the $\mathbf{Z}_0^{\text{consistency}}$ matrix has a -0.5 in the $(1, n+2)$ and $(n+2, 1)$ locations (the negative sign corresponding to the fact that the consistency measure is written as a maximization) and zeros elsewhere. The experiment constraints are similar to the prediction problem, except that terms for the products of the experiment uncertainty and γ are present.

$$\mathbf{Z}_{e,l}^{\text{consistency}} = \begin{bmatrix} l_e + l_{e,\text{fit}} + d_e & \mathbf{0}_{1\times n} & -l_e/2 \\ \mathbf{0}_{n\times 1} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times 1} \\ -l_e/2 & \mathbf{0}_{1\times n} & 0 \end{bmatrix} - \begin{bmatrix} \mathbf{S}_e & \mathbf{0}_{n\times 1} \\ \mathbf{0}_{1\times n} & 0 \end{bmatrix}, \tag{3.15}$$

$$\mathbf{Z}_{e,u}^{\text{consistency}} = \begin{bmatrix} \mathbf{S}_e & \mathbf{0}_{n\times 1} \\ \mathbf{0}_{1\times n} & 0 \end{bmatrix} - \begin{bmatrix} u_e + u_{e,\text{fit}} + d_e & \mathbf{0}_{1\times n} & -u_e/2 \\ \mathbf{0}_{n\times 1} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times 1} \\ -u_e/2 & \mathbf{0}_{1\times n} & 0 \end{bmatrix}. \tag{3.16}$$

In future discussion the $l_{e,\text{fit}}$ and $u_{e,\text{fit}}$ terms will be omitted for notation simplicity. The terms l_e and u_e will notate all experiment and modeling uncertainty, including fitting errors.

The coordinate aligned prior information constraints on the parameters ($\alpha_i \leq x_i \leq \beta_i$ for $i = 1, \ldots, n$) can be formulated as quadratic constraints in many ways. Three options are,

$$(\alpha_i - x_i)(\beta_i - x_i) \leq 0, \tag{3.17}$$

$$\text{or } (\alpha_i - x_i)(\beta_i + \epsilon_i - x_i) \leq 0 \quad \& \quad (\alpha_i - \epsilon_i - x_i)(\beta_i - x_i) \leq 0, \tag{3.18}$$

$$\text{or } \alpha_i - x_i \leq 0 \quad \& \quad x_i - \beta_i \leq 0, \tag{3.19}$$

where ϵ_i in the second line of constraints is some small positive number. Each line of constraints listed constrain x_i to the interval $[\alpha_i, \beta_i]$. However, in the S-procedure formulation the resulting lower bound on the optimal cost can vary. It is proven in [34] that the best lower bound results come from using the constraint in equation (3.17). However, the constraints in equations (3.18) and (3.19) each have the added benefit of their own Lagrange multiplier in the S-procedure formulation. Chapter 4 will show that these Lagrange multipliers can be directly used for sensitivity analysis. If ϵ is chosen sufficiently small, then the lower bound on the optimal cost when using the constraints in equation (3.18) is not much worse than that resulting from the use of the constraints in equation (3.17). Therefore, [34] recommends using the constraints in equation (3.18) with a heuristic choice of $\epsilon_i = 0.05(\beta_i - \alpha_i)$. The resulting two quadratic forms for each parameter are,

$$\mathbf{Z}_{i,\alpha} = \begin{bmatrix} \alpha_i(\beta_i + \epsilon_i) & -(\alpha_i + \beta_i + \epsilon_i)\mathbf{e}_i^{\mathsf{T}}/2 \\ -(\alpha_i + \beta_i + \epsilon_i)\mathbf{e}_i/2 & \mathbf{e}_i\mathbf{e}_i^{\mathsf{T}} \end{bmatrix}, \tag{3.20}$$

$$\mathbf{Z}_{i,\beta} = \begin{bmatrix} \beta_i(\alpha_i - \epsilon_i) & -(\alpha_i + \beta_i - \epsilon_i)\mathbf{e}_i^{\mathsf{T}}/2 \\ -(\alpha_i + \beta_i - \epsilon_i)\mathbf{e}_i/2 & \mathbf{e}_i\mathbf{e}_i^{\mathsf{T}} \end{bmatrix}, \tag{3.21}$$

where $\mathbf{e}_i$ is the i^{th} standard vector, i.e. the vector with a 1 in the i^{th} location and zeros elsewhere. The dimension of $\mathbf{e}_i$ is $n \times 1$ for the prediction problem and $(n+1) \times 1$ for the consistency measure problem.

Chapter 4

Sensitivity Analysis

Sensitivity analysis is a broad range of techniques used to determine the relative importance of inputs to a mathematical or computational model [98]. Typically it involves varying the input parameters to a system to see how the output is affected. The resulting analysis can be used for, among other things, active variable analysis, system reduction, model development and verification [97].

In optimization, Lagrange multipliers, the variables of the Lagrangian dual problem, can be used for sensitivity analysis [92]. Most optimization solvers, automatically solve the dual problem simultaneously, thereby providing sensitivity information for free. These sensitivities provide information about the change in the optimal objective value with respect to changes in the constraint bounds. This is discussed more in Sections 4.1 and 4.2 but we first discuss a simple toy example.

Figure 4.1 shows an example feasible set in two variables made of several linear constraints. The parameters, x_1 and x_2 are constrained to the prior rectangle $\mathcal{H}$ as well as constrained by five experimental constraints labeled $e_1, \ldots, e_5$. The level sets of a model M_0 are shown in light gray. This model only has functional dependence on x_2.

Several interesting things can be seen in Figure 4.1 that might seem counterintuitive. Even though M_0 only depends on x_2, the optimal value is not locally sensitive to the prior bounds of x_2. Similarly, the maximum value of M_0 is dependent on the prior upper bound of x_1. Examples of this type can also be seen with the experiment constraints. The constraint e_1 is redundant and therefore the optimal values have no sensitivity to it, even though it is largely correlated with the gradient direction of M_0. The directions of the constraints of experiments e_4 and e_5 have large components orthogonal to the gradient of M_0, yet the minimum value is sensitive to the bound e_4 and neither optimal value is sensitive to the bound e_5.

This chapter will focus on the sensitivities of the response prediction and consis-

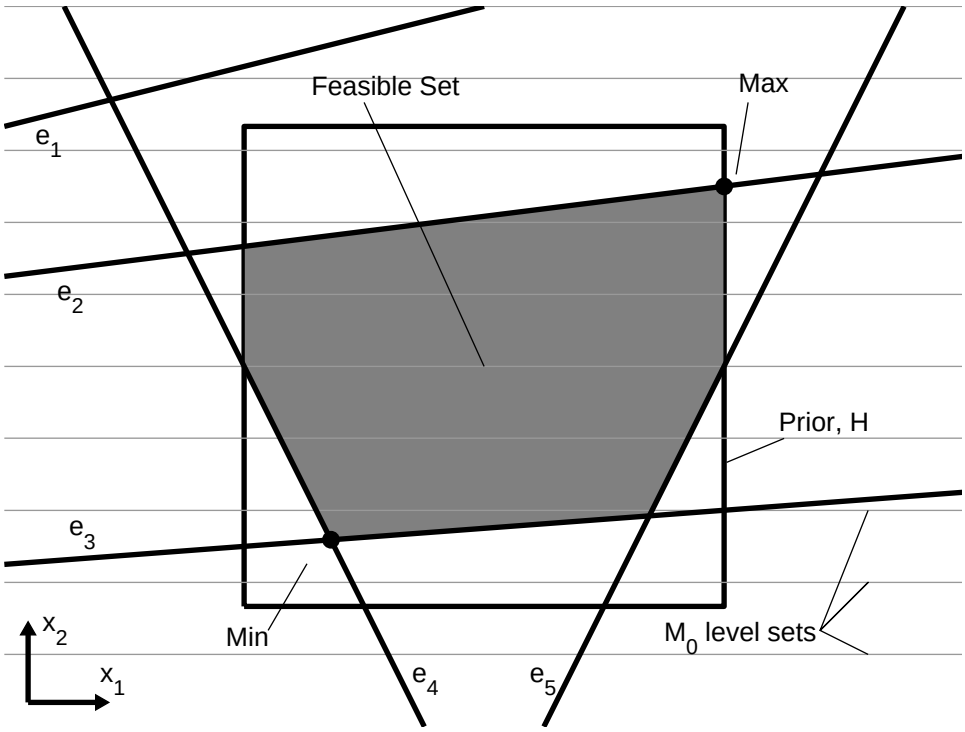

Figure 4.1: Toy example demonstrating various qualitative sensitivities in a response prediction problem. There are two variables x_1 and x_2, each upper and lower bounded (the prior, $\mathcal{H}$). There are five experiment constraints, labeled e_1 through e_5. The feasible set, $\mathcal{F}$ is shaded. The light gray lines represent the level sets of the model to predict, M_0, which only depends on x_2.

tency measure problems. Sections 4.1 and 4.2 will discuss Lagrange multipliers and their interpretation as partial derivatives. Section 4.3 derives the relation between the Lagrange multipliers and the sensitivity to uncertainty in the response prediction and consistency measure problems. Sections 4.6 and 4.7 discuss some uses for the sensitivity information.

4.1 Lagrange Multipliers from the S-procedure

Lagrange multipliers are the basis for the sensitivity analysis presented in this chapter. Lagrange multipliers are the optimal values of the variables from the Lagrange dual optimization problem. The dual problem formulation essentially involves changing the objective function to include a weighted sum of the constraints [14]. Recall from Section 3.1.1 that the S-procedure formulation is such a dual problem formulation.

$$\begin{aligned} \underline{p}_{\mathrm{s}}^{\star} := & \max_{\boldsymbol{\lambda},\gamma} \gamma \\ & \text{s.t.} \begin{cases} \mathbf{Z}_0 - \left[\begin{smallmatrix} \gamma & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + \sum_{i=1}^{N} \lambda_i \mathbf{Z}_i \succeq 0 \\ \boldsymbol{\lambda} \geq 0 \end{cases} \end{aligned} \tag{4.1}$$

The optimal dual variables $\boldsymbol{\lambda}^{\star}$ are Lagrange multipliers for the NQCQP problem. There are two multipliers for each experiment constraint, one each for the upper and lower bounds. Here, the equality constraints in the general NQCQP formulation are omitted, as they are not used for the consistency measure and response prediction problems. The number of multipliers for each parameter depends upon the choice of quadratic formulation of the parameter constraints. As discussed in Section 3.4, we choose a formulation that provides a multiplier for each parameter lower bound and one for each upper bound.

4.2 Lagrange Multipliers as Local Partial Derivatives

The Lagrange multipliers from the outer bound S-procedure formulation are the partial derivatives of the optimal outer bound value with respect to the constraint bounds. To demonstrate this, we first rewrite the optimal NQCQP value as a function of the constraint bounds.

$$\begin{aligned} p^{\star}(\boldsymbol{\xi}) = & \min_{\mathbf{x}} \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_0 \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \\ & \text{s.t.} \ \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \leq \xi_i, \quad i = 1, \ldots, N \end{aligned} \tag{4.2}$$

Note that $p^\star(0)$ is the original solution $p^\star$ as shown in equation (3.1). We can similarly write the S-procedure and rank relaxation bounds as functions of the constraint bounds as $\underline{p}^\star_{\text{s}}(\boldsymbol{\xi})$ and $\underline{p}^\star_{\text{r}}(\boldsymbol{\xi})$. These will be used to describe the Lagrange multipliers. But a definition is needed first.

Definition A function $f : \mathbb{R}^n \to \mathbb{R}$ is called *semi-differentiable* at a point $\mathbf{x} \in \mathbb{R}^n$ if for every direction $\mathbf{u} \in \mathbb{R}^n$ the one sided limit

$$\partial_{\mathbf{u}} f(\mathbf{x}) := \lim_{h \to 0+} \frac{f(\mathbf{x} + h\mathbf{u}) - f(\mathbf{x})}{h} \tag{4.3}$$

exists and is finite [86]. $\partial_{\mathbf{u}} f(\mathbf{x})$ is called the *semi-derivative* with respect to the direction $\mathbf{u}$.

A function of one variable is semi-differentiable at a point if it is left-differentiable and right-differentiable at that point. In other words, the standard limit definition of the derivative exists from the left and the right, though they need not be equal. This means that in the one dimensional case, a semi-differentiable function can have "kinks" in it. The partial semi-derivative of the optimal value of the lower bounds on the NQCQP is used to bound the value of Lagrange multipliers.

Theorem 4.1. *If $\underline{p}^\star_{\text{r}}(\mathbf{0}) = \underline{p}^\star_{\text{s}}(\mathbf{0})$ (strong duality between equation (3.2) and equation (3.4)), and $\underline{p}^\star_{\text{r}}(\boldsymbol{\xi})$ is semi-differentiable at the original constraint bounds $\boldsymbol{\xi} = \mathbf{0}$, then the optimal Lagrange multipliers from the S-procedure formulation, $\boldsymbol{\lambda}^\star$, are bounded by the left and right partial derivatives of the optimal value of the rank relaxation problem, with respect to the constraint bounds. Specifically,*

$$\lambda_i^\star \in \left[-\partial_{\mathbf{e}_i} \underline{p}^\star_{\text{r}}(\mathbf{0}),\ \partial_{-\mathbf{e}_i} \underline{p}^\star_{\text{r}}(\mathbf{0}) \right], \tag{4.4}$$

where $\mathbf{e}_i$ is the i^{th} standard vector, a vector with a one at the i^{th} location and zeros elsewhere.

Proof. Starting with an equality shown in the duality derivation from Chapter 3 we derive a linear lower bound on the optimal outer bound solution.

$$\begin{aligned}
\underline{p}^\star_{\text{s}} &\overset{(a)}{=} \max_{\boldsymbol{\lambda} \geq 0, \gamma} \min_{\mathbf{Q} \succeq 0} \text{Tr}\left[\mathbf{Z}_0 \mathbf{Q}\right] + (1 - Q_{11})\,\gamma + \sum_{i=1}^N \lambda_i \text{Tr}\left[\mathbf{Z}_i \mathbf{Q}\right] \\
&\overset{(b)}{=} \min_{\mathbf{Q} \succeq 0} \text{Tr}\left[\mathbf{Z}_0 \mathbf{Q}\right] + (1 - Q_{11})\,\gamma^\star + \sum_{i=1}^N \lambda_i^\star \text{Tr}\left[\mathbf{Z}_i \mathbf{Q}\right] \\
&\overset{(c)}{\leq} \min_{\mathbf{Q} \succeq 0} \text{Tr}\left[\mathbf{Z}_0 \mathbf{Q}\right] + \sum_{i=1}^N \lambda_i^\star \text{Tr}\left[\mathbf{Z}_i \mathbf{Q}\right] \text{ s.t. } Q_{11} = 1,\ \text{Tr}\left[\mathbf{Z}_i \mathbf{Q}\right] \leq \xi_i \\
&\overset{(d)}{\leq} \min_{\mathbf{Q} \succeq 0} \text{Tr}\left[\mathbf{Z}_0 \mathbf{Q}\right] + \sum_{i=1}^N \lambda_i^\star \xi_i \text{ s.t. } Q_{11} = 1,\ \text{Tr}\left[\mathbf{Z}_i \mathbf{Q}\right] \leq \xi_i \\
&\overset{(e)}{=} \underline{p}^\star_{\text{r}}(\boldsymbol{\xi}) + \sum_{i=1}^N \lambda_i^\star \xi_i
\end{aligned} \tag{4.5}$$

Equality (a) comes from the duality derivation in equation (3.6). By substituting in the optimal values $\boldsymbol{\lambda}^\star$ and $\gamma^\star$ we get equality (b). Adding constraints to the optimization increases the value as in (c). Since $\boldsymbol{\lambda}^\star$ is positive, substituting $\mathrm{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right]$ for the largest constrained value ξ_i gives the upper bound in (d). The final equality (e) uses the definition of definition of $\underline{p}^\star_{\mathrm{r}}$ (see equation (3.4)). This set of relations simplifies to,

$$\underline{p}^\star_{\mathrm{r}}(\boldsymbol{\xi}) \geq \underline{p}^\star_{\mathrm{s}}(0) - \sum_{i=1}^{N} \lambda_i^\star \xi_i. \tag{4.6}$$

The rank reduction optimum, $\underline{p}^\star_{\mathrm{r}}(\boldsymbol{\xi})$ is under bounded by an affine function of the constraint bounds $\boldsymbol{\xi}$. With the hypothesis that strong duality holds between the S-procedure and rank relaxation problems, the bound is rewritten as,

$$\underline{p}^\star_{\mathrm{r}}(\boldsymbol{\xi}) \geq \underline{p}^\star_{\mathrm{r}}(0) - \sum_{i=1}^{N} \lambda_i^\star \xi_i. \tag{4.7}$$

Note that equality holds when $\boldsymbol{\xi} = \mathbf{0}$.

If $\underline{p}^\star_{\mathrm{r}}$ is semi-differentiable at a particular $\boldsymbol{\xi} = \hat{\boldsymbol{\xi}}$, then the left and right partial derivatives exist, are finite, and are written respectively as,

$$\partial_{-\mathbf{e}_i}\underline{p}^\star_{\mathrm{r}}(\hat{\boldsymbol{\xi}}) = \lim_{h\to 0+} \frac{\underline{p}^\star_{\mathrm{r}}(\hat{\boldsymbol{\xi}} - h\mathbf{e}_i) - \underline{p}^\star_{\mathrm{r}}(\hat{\boldsymbol{\xi}})}{h}, \tag{4.8}$$

$$\partial_{\mathbf{e}_i}\underline{p}^\star_{\mathrm{r}}(\hat{\boldsymbol{\xi}}) = \lim_{h\to 0+} \frac{\underline{p}^\star_{\mathrm{r}}(\hat{\boldsymbol{\xi}} + h\mathbf{e}_i) - \underline{p}^\star_{\mathrm{r}}(\hat{\boldsymbol{\xi}})}{h}. \tag{4.9}$$

By hypothesis, semi-differentiability holds at $\boldsymbol{\xi} = \mathbf{0}$. Now a lower bound on the i^{th} Lagrange multiplier is derived by using the inequality in equation (4.7).

$$\begin{aligned}
\partial_{\mathbf{e}_i}\underline{p}^\star_{\mathrm{r}}(\mathbf{0}) &= \lim_{h\to 0+} \frac{\underline{p}^\star_{\mathrm{r}}\left(h\mathbf{e}_i\right) - \underline{p}^\star_{\mathrm{r}}(\mathbf{0})}{h} \\
&\geq \lim_{h\to 0+} \frac{\underline{p}^\star_{\mathrm{r}}(\mathbf{0}) - \lambda_i^\star h - \underline{p}^\star_{\mathrm{r}}(\mathbf{0})}{h} \\
&= -\lambda_i^\star.
\end{aligned} \tag{4.10}$$

And similarly, an upper bound is derived.

$$\begin{aligned}
\partial_{-\mathbf{e}_i}\underline{p}^\star_{\mathrm{r}}(\mathbf{0}) &= \lim_{h\to 0+} \frac{\underline{p}^\star_{\mathrm{r}}\left(-h\mathbf{e}_i\right) - \underline{p}^\star_{\mathrm{r}}(\mathbf{0})}{h} \\
&\geq \lim_{h\to 0+} \frac{\underline{p}^\star_{\mathrm{r}}(\mathbf{0}) + \lambda_i^\star h - \underline{p}^\star_{\mathrm{r}}(\mathbf{0})}{h} \\
&= \lambda_i^\star.
\end{aligned} \tag{4.11}$$

Thus the desired left and right bounds on $\lambda_i^\star$ are achieved. □

If differentiability is assumed in the previous theorem, an even stronger result follows.

Corollary 4.2. *If $\underline{p}^\star_{\mathrm{r}}(\mathbf{0}) = \underline{p}^\star_{\mathrm{s}}(\mathbf{0})$, and $\underline{p}^\star_{\mathrm{r}}(\boldsymbol{\xi})$ is differentiable at the original constraint bounds $\boldsymbol{\xi} = \mathbf{0}$, then the optimal Lagrange multipliers from the S-procedure formulation, $\boldsymbol{\lambda}^\star$, are the negative partial derivatives of the optimal value of the rank relaxation problem, with respect to the constraint bounds. Specifically,*

$$\lambda_i^\star = - \left. \frac{\partial \underline{p}^\star_{\mathrm{r}}}{\partial \xi_i} \right|_{\boldsymbol{\xi}=\mathbf{0}} \tag{4.12}$$

Proof. If the optimal lower bound is differentiable at $\boldsymbol{\xi} = \mathbf{0}$ then the bounds on $\lambda_i^\star$ in Theorem 4.1 are equal. □

Furthermore, with stronger conditions, the Lagrange multipliers provide direct sensitivities to the original problem.

Theorem 4.3. *If strong duality holds between the original NQCQP and the S-procedure solution, $p^\star(\mathbf{0}) = \underline{p}^\star_{\mathrm{s}}(\mathbf{0})$, and $p^\star(\boldsymbol{\xi})$ is differentiable at the original constraint bounds $\boldsymbol{\xi} = \mathbf{0}$, then the optimal Lagrange multipliers from the S-procedure formulation, $\boldsymbol{\lambda}^\star$, are the negative partial derivatives of the optimal value of the primal NQCQP with respect to the constraint bounds. Specifically,*

$$\lambda_i^\star = - \left. \frac{\partial p^\star}{\partial \xi_i} \right|_{\boldsymbol{\xi}=\mathbf{0}} \tag{4.13}$$

Proof. Chapter 3 demonstrated that

$$p^\star(\boldsymbol{\xi}) \geq \underline{p}^\star_{\mathrm{r}}(\boldsymbol{\xi}) \geq \underline{p}^\star_{\mathrm{s}}(\boldsymbol{\xi}), \ \forall \boldsymbol{\xi}. \tag{4.14}$$

Furthermore, the proof of Theorem 4.1 shows the relation in equation (4.6). Therefore,

$$p^\star(\boldsymbol{\xi}) \geq \underline{p}^\star_{\mathrm{s}}(0) - \sum_{i=1}^{N} \lambda_i^\star \xi_i. \tag{4.15}$$

From here the proof is the same as that of Theorem 4.1 and Corollary 4.2. □

It has been our experience that the conditions for Corollary 4.2 usually hold and the conditions for Theorem 4.3 often do not (except for simple problems). Through the rest of this chapter, we assume that the conditions for the corollary hold, and therefore the Lagrange multipliers are the exact partial derivatives shown in equation (4.12).

4.3 Sensitivity to Experiment and Parameter Uncertainty

Typically, sensitivity analysis examines the "by-value" sensitivities. These are the sensitivities of a model's output value to specific values of the input parameters. The sensitivities provided by the Lagrange multipliers, as discussed in the previous sections, provide sensitivities of the optimal value of the optimization problem with respect to the constraint bounds. The constraints in the Data Collaboration formulation are bounded by the uncertainty in the parameters and experiment data. Therefore, in past work, we have described the Lagrange multipliers from the solution of Data Collaboration problems as "by-uncertainty" sensitivity [95].

For each type of question or problem formulated in the Data Collaboration framework, the sensitivity information might be interpreted slightly differently. Therefore, we focus on the response prediction and consistency measure problems. Throughout this section we assume the conditions for Corollary 4.2 hold.

4.3.1 Response Prediction Sensitivity

Once the various experiment models are fitted with quadratic surrogate approximations, the response prediction problem is simply two NQCQPs: a minimization for the lower bound L_0 and a maximization for the upper bound R_0. Each experiment constraint has two inequalities, one for the lower uncertainty and one for the upper uncertainty. The Lagrange multipliers from the S-procedure formulation for this constraint are denoted $\underline{\lambda}^\star_{e,u}$ and $\overline{\lambda}^\star_{e,u}$, where the underline denotes the multiplier associated with the lower bound prediction L_0 and the overline denotes that associated with the upper bound prediction.

In the rank relaxation formulation, the upper constraint on experiment e is,

$$\text{Tr}\left[\mathbf{Z}_{e,u}\mathbf{Q}\right] = \text{Tr}\left[\left(\mathbf{S}_e - \begin{bmatrix} u_e + d_e & \mathbf{0}_{1\times n} \\ \mathbf{0}_{n\times 1} & \mathbf{0}_{n\times n}\end{bmatrix}\right)\mathbf{Q}\right] \leq \xi_{e,u}, \tag{4.16}$$

We are interested in the relationship between a change in the uncertainty u_e and the change in the bound $\xi_{e,u}$. In this case, the relation is unitary. Specifically, for the optimal $\mathbf{Q}$, $\partial\xi_{e,u}/\partial u_e = Q^\star_{11} = 1$. Given these relations we can derive the local sensitivity of the prediction bound to the uncertainty. For example,

$$\begin{aligned}\left.\frac{\partial \underline{L}_0}{\partial u_e}\right|_{u_e=\hat{u}_e} &= \left.\frac{\partial \underline{L}_0}{\partial \xi_{e,u}}\right|_{\boldsymbol{\xi}=\mathbf{0}} \cdot \left.\frac{\partial \xi_{e,u}}{\partial u_e}\right|_{u_e=\hat{u}_e} \\ &= (-\underline{\lambda}^\star_{e,u})\cdot(1).\end{aligned} \tag{4.17}$$

Here, u_e is meant to denote the uncertainty as a variable, and $\hat{u}_e$ is the nominal value of the uncertainty. This derivation is similar for all the experiment sensitivities

associated with response prediction, with only a few sign changes involved. The four types of sensitivities associated with the experiments are

$$\left.\frac{\partial \underline{L}_0}{\partial l_e}\right|_{l_e=\hat{l}_e} = \underline{\lambda}^{\star}_{e,l}, \tag{4.18a}$$

$$\left.\frac{\partial \underline{L}_0}{\partial u_e}\right|_{u_e=\hat{u}_e} = -\underline{\lambda}^{\star}_{e,u}, \tag{4.18b}$$

$$\left.\frac{\partial \overline{R}_0}{\partial l_e}\right|_{l_e=\hat{l}_e} = -\overline{\lambda}^{\star}_{e,l}, \tag{4.18c}$$

$$\left.\frac{\partial \overline{R}_0}{\partial u_e}\right|_{u_e=\hat{u}_e} = \overline{\lambda}^{\star}_{e,u}. \tag{4.18d}$$

The sensitivities associated with the parameter constraints are derived in a similar manner. For example, for the upper-bound constraint discussed in Section 3.4, the rank relaxation formulation constraint is,

$$\mathrm{Tr}\left[\mathbf{Z}_{i,\beta}\mathbf{Q}\right] = \mathrm{Tr}\left[\left(\begin{bmatrix} \beta_i(\alpha_i-\epsilon_i) & -(\alpha_i+\beta_i-\epsilon_i)\mathbf{e}_i^{\mathsf{T}}/2 \\ -(\alpha_i+\beta_i+\epsilon_i)\mathbf{e}_i/2 & \mathbf{e}_i\mathbf{e}_i^{\mathsf{T}} \end{bmatrix}\right)\mathbf{Q}\right] \leq \xi_{i,\beta}. \tag{4.19}$$

At the optimal $\mathbf{Q}^{\star,L}$ for the left endpoint $\underline{L}_0$, the derivative of $\xi_{i,\beta}$ with respect to β_i is

$$\begin{aligned} \frac{\partial \xi_{i,\beta}}{\partial \beta_i} &= Q^{\star,L}_{(1,i+1)} - (\alpha_i-\epsilon_i)Q^{\star,L}_{(1,1)} \\ &= Q^{\star,L}_{(1,i+1)} - \alpha_i + \epsilon_i. \end{aligned} \tag{4.20}$$

The Lagrange multipliers from the S-procedure formulation associated with the upper bound of the i^{th} constraint are denoted as $\underline{\nu}^{\star}_{i,\beta}$ and $\overline{\nu}^{\star}_{i,\beta}$, where the underline and overline refer to the left and right prediction bounds, respectively. The local sensitivity of the left prediction bound can now be derived as,

$$\begin{aligned} \left.\frac{\partial \underline{L}_0}{\partial \beta_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} &= \left.\frac{\partial \underline{L}_0}{\partial \xi_{i,\beta}}\right|_{\boldsymbol{\xi}=\mathbf{0}} \cdot \left.\frac{\partial \xi_{i,\beta}}{\partial \beta_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} \\ &= \left(-\nu^{\star}_{i,\beta}\right) \cdot \left(Q^{\star,L}_{(1,i+1)} - \hat{\alpha}_i + \epsilon_i\right). \end{aligned} \tag{4.21}$$

The sensitivity due to the lower bound on the parameter, and the sensitivities of the right prediction bound due to both parameter bounds are similarly derived. The

resulting sensitivities are,

$$\left.\frac{\partial \underline{L}_0}{\partial \alpha_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} = -\underline{\nu}^{\star}_{i,\alpha} \cdot \left(Q^{\star,L}_{(1,i+1)} - \hat{\beta}_i - \epsilon_i\right), \tag{4.22a}$$

$$\left.\frac{\partial \underline{L}_0}{\partial \beta_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} = -\underline{\nu}^{\star}_{i,\beta} \cdot \left(Q^{\star,L}_{(1,i+1)} - \hat{\alpha}_i + \epsilon_i\right), \tag{4.22b}$$

$$\left.\frac{\partial \overline{R}_0}{\partial \alpha_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} = \overline{\nu}^{\star}_{i,\alpha} \cdot \left(Q^{\star,R}_{(1,i+1)} - \hat{\beta}_i - \epsilon_i\right), \tag{4.22c}$$

$$\left.\frac{\partial \overline{R}_0}{\partial \beta_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} = \overline{\nu}^{\star}_{i,\beta} \cdot \left(Q^{\star,R}_{(1,i+1)} - \hat{\alpha}_i + \epsilon_i\right). \tag{4.22d}$$

4.3.2 Consistency Measure Sensitivity

The consistency measure problem has an extra variable γ that plays a role in the experiment constraints. Thus the sensitivity derivations associated with the experiment constraints are slightly different than those for the prediction problem. In the rank relaxation formulation, the upper constraint on experiment e is,

$$\mathrm{Tr}\left[\mathbf{Z}_{e,u}\mathbf{Q}\right] = \mathrm{Tr}\left[\left(\begin{bmatrix} \mathbf{S}_e & \mathbf{0}_{n\times 1} \\ \mathbf{0}_{1\times n} & 0 \end{bmatrix} - \begin{bmatrix} u_e + u_{e,\mathrm{fit}} + d_e & \mathbf{0}_{1\times n} & -u_e/2 \\ \mathbf{0}_{n\times 1} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times 1} \\ -u_e/2 & \mathbf{0}_{1\times n} & 0 \end{bmatrix}\right)\mathbf{Q}\right] \leq \xi_{e,u}. \tag{4.23}$$

At the optimal $\mathbf{Q}^{\star}$, the derivative of $\xi_{e,u}$ with respect to u_e is

$$\begin{aligned} \frac{\partial \xi_{e,u}}{\partial u_e} &= Q^{\star}_{(1,1)} - Q^{\star}_{(1,n+2)} \\ &= 1 - \overline{\mathrm{C}}_{\mathcal{D}}. \end{aligned} \tag{4.24}$$

The optimal Lagrange multipliers for the consistency problem associated with the left and right experiment bounds are denoted $\lambda^{\star}_{e,l}$ and $\lambda^{\star}_{e,u}$ respectively. The sensitivities of the consistency measure with respect to the experiment constraints are the same as before.

$$\begin{aligned} \left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial u_e}\right|_{u_e=\hat{u}_e} &= \left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial \xi_{e,u}}\right|_{\boldsymbol{\xi}=\mathbf{0}} \cdot \left.\frac{\partial \xi_{e,u}}{\partial u_e}\right|_{u_e=\hat{u}_e}, \\ &= \lambda^{\star}_{e,u} \cdot \left(1 - \overline{\mathrm{C}}_{\mathcal{D}}(u_e = \hat{u}_e)\right). \end{aligned} \tag{4.25}$$

The multiplier $\lambda^{\star}_{e,u}$ enters positively because the consistency measure is a maximization problem. Since $\overline{\mathrm{C}}_{\mathcal{D}}$ is thought of as a function of constraint bounds, $\overline{\mathrm{C}}_{\mathcal{D}}(u_e = \hat{u}_e)$ refers to the consistency measure at the "current" constraint bounds. Naturally, it is a function of all of the constraint bounds, not just u_e, but we leave them out here for notational simplicity.

The sensitivity of the consistency measure due to the left bound is derived similarly. To summarize, both types of experiment sensitivities are expressed as,

$$\left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial l_e}\right|_{l_e=\hat{l}_e} = -\lambda^{\star}_{e,l} \cdot \left(1 - \overline{\mathrm{C}}_{\mathcal{D}}(l_e = \hat{l}_e)\right), \tag{4.26a}$$

$$\left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial u_e}\right|_{u_e=\hat{u}_e} = \lambda^{\star}_{e,u} \cdot \left(1 - \overline{\mathrm{C}}_{\mathcal{D}}(u_e = \hat{u}_e)\right). \tag{4.26b}$$

The parameter bound sensitivities are derived exactly the same as in the previous section since the parameter constraints have no dependence on the variable γ. They are expressed as,

$$\left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial \alpha_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} = \nu^{\star}_{i,\alpha} \cdot \left(Q^{\star}_{(1,i+1)} - \hat{\beta}_i - \epsilon_i\right), \tag{4.27a}$$

$$\left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial \beta_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} = \nu^{\star}_{i,\beta} \cdot \left(Q^{\star}_{(1,i+1)} - \hat{\alpha}_i + \epsilon_i\right), \tag{4.27b}$$

where $\nu^{\star}_{i,\alpha}$ and $\nu^{\star}_{i,\beta}$ are the Lagrange multipliers for the consistency measure problem associated with the i^{th} lower and upper parameter bounds respectively.

4.3.3 Sensitivity Pairs

Every experiment and parameter has two constraints, a lower and an upper bound. Assuming that each left bound is strictly less than the corresponding right bound for each of these constraints, it is intuitive that the consistency measure and prediction bounds are not sensitive to both bounds. The optimization problem is only sensitive to a bound if the optimal $\mathbf{x}^{\star}$ is on that boundary. Naturally, if the boundaries are separated, $\mathbf{x}^{\star}$ cannot be on both the upper and lower boundaries simultaneously. Therefore, for each pair of sensitivities of the primal problem to the constraint bounds, at least one of them is zero.

This section will demonstrate that this intuition holds even for the sensitivities derived from the S-procedure formulations for the experiment constraints. This is asserted in two theorems, one for the response prediction case, and one for the consistency measure case.

Theorem 4.4. *For each experiment constraint e with nonzero uncertainty ($l_e \neq u_e$), at least one of the sensitivities of the outer left prediction bound $\underline{L}_0$ with respect to l_e and u_e is zero. In other words, at most one of $\underline{\lambda}^{\star}_{e,l}$ and $\underline{\lambda}^{\star}_{e,u}$ can be nonzero. The same is true for the sensitivities associated with the outer right prediction $\overline{R}_0$.*

Proof. Without loss of generality, we focus on the left prediction bound problem. Assume that for a specific experiment e, the optimal $\underline{\lambda}^{\star}_{e,l}$ and $\underline{\lambda}^{\star}_{e,u}$ are both strictly

positive. We will show a contradiction. Assuming a consistent dataset, the optimal Lagrange multipliers $\underline{\boldsymbol{\lambda}}^\star$ and maximal $\gamma^\star$ satisfy the semi-definite constraint

$$\mathbf{Z}_0 - \left[\begin{smallmatrix} \gamma^\star & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + \sum_{i=1}^{N} \underline{\lambda}_i \mathbf{Z}_i \succeq 0. \tag{4.28}$$

Examine the part of this summation associated $\gamma^\star$ and $\lambda^\star_{e,l}$ and $\lambda^\star_{e,u}$. Specifically,

$$\begin{aligned} &\underline{\lambda}^\star_{e,l}\mathbf{Z}_{e,l} + \underline{\lambda}^\star_{e,u}\mathbf{Z}_{e,u} - \left[\begin{smallmatrix} \gamma^\star & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] \\ =&\underline{\lambda}^\star_{e,l}\left[\begin{smallmatrix} d_e+l_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] - \underline{\lambda}^\star_{e,u}\left[\begin{smallmatrix} d_e+u_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + (\underline{\lambda}^\star_{e,u} - \underline{\lambda}^\star_{e,l})\mathbf{S}_e - \left[\begin{smallmatrix} \gamma^\star & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right]. \end{aligned} \tag{4.29}$$

Now define new variables,

$$\begin{aligned} \tilde{\lambda}_{e,l} &:= \underline{\lambda}^\star_{e,l} - \min\{\underline{\lambda}^\star_{e,l}, \underline{\lambda}^\star_{e,u}\} \\ \tilde{\lambda}_{e,u} &:= \underline{\lambda}^\star_{e,u} - \min\{\underline{\lambda}^\star_{e,l}, \underline{\lambda}^\star_{e,u}\} \\ \tilde{\gamma} &:= \gamma^\star + (u_e - l_e)\min\{\underline{\lambda}^\star_{e,l}, \underline{\lambda}^\star_{e,u}\}. \end{aligned} \tag{4.30}$$

Both $\tilde{\lambda}_{e,l}$ and $\tilde{\lambda}_{e,u}$ are nonnegative, and at least one of them is zero. Furthermore, the partial sum (4.29) remains unchanged.

$$\begin{aligned} &\tilde{\lambda}_{e,l}\left[\begin{smallmatrix} d_e+l_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] - \tilde{\lambda}_{e,u}\left[\begin{smallmatrix} d_e+u_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + (\tilde{\lambda}_{e,u} - \tilde{\lambda}_{e,l})\mathbf{S}_e - \left[\begin{smallmatrix} \tilde{\gamma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] \\ =&\underline{\lambda}^\star_{e,l}\left[\begin{smallmatrix} d_e+l_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] - \underline{\lambda}^\star_{e,u}\left[\begin{smallmatrix} d_e+u_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + (\underline{\lambda}^\star_{e,u} - \underline{\lambda}^\star_{e,l})\mathbf{S}_e - \left[\begin{smallmatrix} \gamma^\star & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right]. \end{aligned} \tag{4.31}$$

Therefore, the semi-definite constraint (4.28) is satisfied with $\tilde{\lambda}_{e,l}$, $\tilde{\lambda}_{e,u}$, and $\tilde{\gamma}$ instead of $\underline{\lambda}^\star_{e,l}$, $\underline{\lambda}^\star_{e,u}$, and $\gamma^\star$, respectively. Lastly note,

$$\tilde{\gamma} = \gamma^\star + (u_e - l_e)\min\{\underline{\lambda}^\star_{e,l}, \underline{\lambda}^\star_{e,u}\} \geq \gamma^\star. \tag{4.32}$$

Therefore, $\underline{\lambda}^\star_{e,l}$, $\underline{\lambda}^\star_{e,u}$, and $\gamma^\star$ are not optimal. Contradiction. □

Theorem 4.5. *If the optimal upper bound on the consistency measure from the rank relaxation formulation is differentiable with respect to the constraint bounds, then for each experiment constraint e with nonzero uncertainty ($l_e \neq u_e$), at least one of the sensitivities of the upper bound on the consistency measure $\overline{\mathrm{C}}_{\mathcal{D}}$ with respect to l_e and u_e is zero. In other words, at most one of $\partial\overline{\mathrm{C}}_{\mathcal{D}}/\partial l_e$ and $\partial\overline{\mathrm{C}}_{\mathcal{D}}/\partial u_e$, evaluated at the original bounds $l_e = \hat{l}_e$ and $u_e = \hat{u}_e$, can be nonzero.*

Before proving this, we prove a lemma relating inactive constraints to zero-valued multipliers.

Lemma 4.6. *If the optimal $\mathbf{Q}^\star$ for the rank relaxation lower bound of an NQCQP strictly satisfies one of the inequality constraints, i.e. $\mathrm{Tr}\,[\mathbf{Z}_{\hat{\imath}}\mathbf{Q}^\star] < 0$ for some $\hat{\imath} \in \{1, \ldots, N\}$, and if the optimal solution as a function of the constraint bounds, $\underline{p}^\star_{\mathrm{r}}(\boldsymbol{\xi})$, is differentiable at $\boldsymbol{\xi} = \mathbf{0}$, then $\lambda^\star_{\hat{\imath}} = 0$.*

Proof. Recall from Chapter 3 that we rewrote the rank relaxation problem as a function of its constraint bounds.

$$\begin{aligned} \underline{p}^\star_{\text{r}}(\boldsymbol{\xi}) = &\min_{\mathbf{Q}\succeq 0} \ \text{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] \\ &\text{s.t. } \text{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] \leq \xi_i, \ i = 1, \ldots, N \end{aligned} \tag{4.33}$$

Without loss of generality, the possible equality constraints are not considered for notational simplicity. The feasible set of the original problem is

$$F = \{\mathbf{Q} \succeq 0 \ : \ \text{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] \leq 0, \ i = 1, \ldots, N\}. \tag{4.34}$$

For any $\varepsilon > 0$ denote the feasible set associated with $\underline{p}^\star_{\text{r}}(\varepsilon e_{\hat{\imath}})$ is

$$F_\varepsilon = \{\mathbf{Q} \succeq 0 \ : \ \text{Tr}\left[\mathbf{Z}_{\hat{\imath}}\mathbf{Q}\right] \leq -\varepsilon, \ \text{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] \leq 0, \ i = 1, \ldots, \hat{\imath}-1, \hat{\imath}+1, \ldots, N\}. \tag{4.35}$$

Then $F_\varepsilon \subseteq F$ for all $\varepsilon > 0$. Let $\mathbf{Q}^\star$ be the optimal matrix for the rank relaxation problem at $\boldsymbol{\xi} = \mathbf{0}$ such that $\underline{p}^\star_{\text{r}}(\mathbf{0}) = \text{Tr}\left[\mathbf{Z}_0\mathbf{Q}^\star\right]$. By the lemma's hypothesis, there is an $\hat{\imath} \in \{1, \ldots, N\}$ such that $\text{Tr}\left[\mathbf{Z}_{\hat{\imath}}\mathbf{Q}^\star\right] < 0$. Then there exists $\epsilon > 0$ such that for all $\epsilon' \leq \epsilon$, $\text{Tr}\left[\mathbf{Z}_{\hat{\imath}}\mathbf{Q}^\star\right] \leq -\epsilon'$. Therefore, for all $\epsilon' \leq \epsilon$, $\mathbf{Q}^\star \in F_{\epsilon'}$ and hence $\underline{p}^\star_{\text{r}}(\mathbf{0}) = \underline{p}^\star_{\text{r}}(-\epsilon' e_{\hat{\imath}})$. Then the optimal Lagrange multiplier associated with the $\hat{\imath}^{\text{th}}$ constraint is

$$\lambda^\star_{\hat{\imath}} = \left.\frac{\partial \underline{p}^\star_{\text{r}}}{\partial \xi_{\hat{\imath}}}\right|_{\boldsymbol{\xi}=\mathbf{0}} = \lim_{h\to 0} \frac{\underline{p}^\star_{\text{r}}(\mathbf{0}) - \underline{p}^\star_{\text{r}}(-\epsilon' e_{\hat{\imath}})}{h} = 0. \tag{4.36}$$

□

Proof of Theorem 4.5. If $\overline{\text{C}}_{\mathcal{D}} = 1$, equation (4.27) shows that the partial derivatives $\partial\overline{\text{C}}_{\mathcal{D}}/\partial l_e$ and $\partial\overline{\text{C}}_{\mathcal{D}}/\partial l_e$ at the initial uncertainty will both be zero.

Assume, now, that $\overline{\text{C}}_{\mathcal{D}} < 1$. We assume that both partial derivatives are positive and show a contradiction. If both

$$\left.\frac{\partial\overline{\text{C}}_{\mathcal{D}}}{\partial l_e}\right|_{l_e=\hat{l}_e} > 0, \text{ and } \left.\frac{\partial\overline{\text{C}}_{\mathcal{D}}}{\partial u_e}\right|_{u_e=\hat{u}_e} > 0 \tag{4.37}$$

and since $\overline{\text{C}}_{\mathcal{D}} < 1$, $\lambda^\star_{e,l} > 0$ and $\lambda^\star_{e,u} > 0$. From the contrapositive implication of Lemma 4.6 $\mathbf{Q}^\star$ is on the respective constraint boundaries such that

$$\text{Tr}\left[\mathbf{Z}_{l,\text{consis}}\mathbf{Q}\right] = 0, \text{ and } \text{Tr}\left[\mathbf{Z}_{u,\text{consis}}\mathbf{Q}\right] = 0. \tag{4.38}$$

Let $\hat{\mathbf{Q}}^\star$ be the matrix corresponding to the upper-left $n+1 \times n+1$ submatrix of $\mathbf{Q}^\star$. Then the equalities in equation (4.38) can be written as

$$l_e + d_e - l_e Q_{(1,n+2)} - \text{Tr}\left[\mathbf{S}_e\hat{\mathbf{Q}}^\star\right] = 0, \text{ and } \text{Tr}\left[\mathbf{S}_e\hat{\mathbf{Q}}^\star\right] - u_e - d_e + u_e Q_{(1,n+2)} = 0. \tag{4.39}$$

The consistency measure upper bound $\overline{\mathrm{C}}_{\mathcal{D}}$ is equal to $Q_{(1,n+2)}$. Summing the two equalities then yields,

$$\begin{aligned}\left(l_e + d_e - l_e\overline{\mathrm{C}}_{\mathcal{D}} - \mathrm{Tr}\left[\mathbf{S}_e\hat{\mathbf{Q}}^\star\right]\right) + \left(\mathrm{Tr}\left[\mathbf{S}_e\hat{\mathbf{Q}}^\star\right] - u_e - d_e + u_e\overline{\mathrm{C}}_{\mathcal{D}}\right) = 0 \\ \Rightarrow \quad l_e - u_e - l_e\overline{\mathrm{C}}_{\mathcal{D}} + u_e\overline{\mathrm{C}}_{\mathcal{D}} = 0 \\ \Rightarrow \quad (u_e - l_e)\overline{\mathrm{C}}_{\mathcal{D}} = u_e - l_e \\ \Rightarrow \quad \overline{\mathrm{C}}_{\mathcal{D}} = 1.\end{aligned} \tag{4.40}$$

Contradiction.

□

As mentioned above, a similar theorem for parameter constraints does not always hold for the parameter constraints in the outer bound formulation when they are formed as a pair of quadratic constraints as demonstrated in equation (3.21) on page 18. From experience, it is almost always true that at least one of the two Langrangian multipliers associated with the bounds of a parameter is zero. We now demonstrate a simple example, where both multipliers are nonzero.

Suppose we wish to make a response prediction of the function $y = 1 - x^2$ where the parameter x is a scalar and subject to the bounds $-1 \leq x \leq 1$. No experiment constraints are imposed. The S-procedure formulation of the left bound on the prediction is

$$\begin{aligned}&\max_{\boldsymbol{\lambda}\geq 0,\gamma} \quad \gamma \\ &\text{s.t.} \ \left[\begin{smallmatrix}1 & 0\\ 0 & -1\end{smallmatrix}\right] - \left[\begin{smallmatrix}\gamma & 0\\ 0 & 0\end{smallmatrix}\right] + \lambda_1\left[\begin{smallmatrix}-(1+\epsilon) & -\epsilon/2\\ -\epsilon/2 & 1\end{smallmatrix}\right] + \lambda_1\left[\begin{smallmatrix}-(1+\epsilon) & \epsilon/2\\ \epsilon/2 & 1\end{smallmatrix}\right] \succeq 0.\end{aligned} \tag{4.41}$$

The optimal Lagrange multipliers are $\lambda_1^\star = \lambda_2^\star = 0.5 > 0$.

While the rank relaxation outer bound may be sensitive to both parameter bounds (for a single parameter), this does not mean that actual optimal value is. In fact, it will not be. In the case of strong duality between the rank relaxation problem and the original problem, one of the pair of the Lagrange multipliers associated with a parameter will be zero. If the conditions for Theorem 4.3 hold, then the the multipliers are the partial derivatives of the optimal value of the original problem with respect to the constraints. In this problem, both constraints cannot be zero simultaneously, hence the multipliers can not both be strictly positive (see Lemma 4.6).

4.4 Variable and Output Transformations

To exploit semidefinite programming optimization techniques, transformations to both the inputs (parameters) and outputs (responses) of an observable's model are employed with the goal of obtaining an input/output relation that is well- approximated by a quadratic function.

Suppose that the parameters are denoted by a parameter vector $\mathbf{k}$, the rectangular set $\mathcal{P} = \{\mathbf{k} : \underline{k}_i \leq k_i \leq \overline{k}_i\}$ representing the prior information, and the models by G_e. Associated with each parameter k_i there is a nondecreasing invertible transformation, ϕ_i, which is used to define a new variable,

$$x_i = \phi_i(k_i), \tag{4.42}$$

and its bounds

$$\alpha_i = \phi_i(\underline{k}_i), \quad \beta_i = \phi_i(\overline{k}_i). \tag{4.43}$$

Since the transformation is invertible, it follows that $k_i = \phi_i^{-1}(x_i)$. For a parameter vector $\mathbf{k}$, write $\mathbf{x} = \Phi(\mathbf{k})$ to denote the element-by-element transformation into the vector $\mathbf{x}$. The prior bound $\mathcal{H}$, as previously defined, can now be written as the rectangle $\{\Phi(\mathbf{k}) : k_i \in \mathcal{P}\}$, which is the prior information in the new coordinates, $\mathbf{x}$. Similarly, associated with each model, there is an invertible output transformation ψ_e.

Together, the variable and output transformations allow redefinition of the models, via,

$$M_e(\mathbf{x}) = \psi_e\left(G_e\left(\Phi^{-1}(x)\right)\right). \tag{4.44}$$

The choice of transformations is guided by the goal that each M_e should be well approximated by a quadratic function over the domain $\mathbf{x} \in \mathcal{H}$. The lower and upper experiment uncertainty bounds, $\mathfrak{l}_e$ and $\mathfrak{u}_e$, and the measured value $\mathfrak{d}_e$ are also transformed using the same function as,

$$d_e = \psi_e(\mathfrak{d}_e), \quad l_e = \psi_e(\mathfrak{l}_e), \quad u_e = \psi_e(\mathfrak{u}_e) \tag{4.45}$$

Consider the minimization problem to solve the response prediction left endpoint, L_0. Assuming ψ_0 is monotone and invertible, then the left endpoint of G_0, denoted $\mathfrak{L}_0$, is $\psi_0^{-1}(L_0)$. Consider the partial derivative of the lower bound prediction of a model G_0 with respect to the perturbation in the i^{th} parameter bound. To convert this derivative to be with respect to the original parameter upper bound $\overline{k}_i$ we use the chain rule. Then, for example, the sensitivity of the left prediction bound to the parameter upper bound in the original units is,

$$\left.\frac{\partial \mathfrak{L}_0}{\partial k_i}\right|_{k_i=\hat{k}_i} = \left.\frac{\partial \psi_0^{-1}}{\partial \underline{L}_0}\right|_{\underline{L}_0=\hat{\underline{L}}_0} \cdot \left.\frac{\partial \underline{L}_0}{\partial \beta_i}\right|_{\boldsymbol{\beta}=\hat{\boldsymbol{\beta}}} \cdot \left.\frac{\partial \phi_i}{\partial k_i}\right|_{k_i=\hat{k}_i}. \tag{4.46}$$

Sensitivities with respect to the experiment uncertainty are derived similarly. For example, the sensitivity of the left prediction bound of the upper experiment uncertainty in the original units is,

$$\left.\frac{\partial \mathfrak{L}_0}{\partial \mathfrak{u}_e}\right|_{\mathfrak{u}_e=\hat{\mathfrak{u}}_e} = \left.\frac{\partial \psi_0^{-1}}{\partial \underline{L}_0}\right|_{\underline{L}_0=\hat{\underline{L}}_0} \cdot \left.\frac{\partial \underline{L}_0}{\partial u_e}\right|_{u_e=\hat{u}_e} \cdot \left.\frac{\partial \psi_e}{\partial \mathfrak{u}_e}\right|_{\mathfrak{u}_e=\hat{\mathfrak{u}}_e}. \tag{4.47}$$

Using the transformation in equations (4.46) and (4.47), sensitivities in a problem's original variables are available. The next section uses such transformations to analyze results from the GRI-Mech 3.0 dataset.

4.5 Example: GRI-Mech 3.0

Section 2.6 presented the GRI-Mech 3.0 dataset, a collection of process models and data related to methane combustion. There are 102 parameters (mostly constants associated with reaction rates), and 77 experiments with quadratic models and experimental data. Experimental uncertainty information was not available, and in Section 2.6 and previous work [32, 33, 95] an arbitrary uncertainty of 0.1 was used for all experiments.

We now consider a set of experiment uncertainties provided by a domain expert [120, 121]. These uncertainties are chosen experiment by experiment to be reasonable values for the experimental type and setup. We will use these uncertainties throughout the rest of this document when using the GRI- Mech 3.0 dataset. Since the uncertainties were not provided by the experimenters themselves, this dataset is used only to demonstrate the technique and not to reach any conclusions about the models or experiments therein.

The consistency measure bounds were computed as $[-0.37, -0.26]$ with no branch and bound iterations. This means that every experiment uncertainty must be tightened by at least 26% in order to find a feasible parameter vector and tightening them by more than 37% guarantees the existance of a feasible point. Therefore the dataset is inconsistent. The sensitivity measures can provide insight into what might be causing this inconsistency.

Using the notation from Section 4.4, each of the GRI-Mech 3.0 parameters lies in the rectangular set $\mathcal{P}$; each parameter k_i has an upper and lower bound.

$$\underline{k}_i \leq k_i \leq \overline{k}_i. \tag{4.48}$$

We define the *span* s_i of k_i as $\sqrt{\overline{k}_i/\underline{k}_i}$, and the following transformation,

$$x_i = \phi_i(k_i) = \frac{\log_{10}(k_i) - \log_{10}\left(\sqrt{\underline{k}_i \cdot \overline{k}_i}\right)}{\log_{10}(s_i)}. \tag{4.49}$$

This transformation normalizes the bounds of x_i to be $[-1, 1]$. For each experiment we also have ψ_e which is just a simple $\log_{10}$ transformation. However, for this type of problem, kineticists are interested in the sensitivity with respect to the log of the parameter bound $\overline{k}_i$ and with respect to the log of the experiment bound $\mathfrak{u}_e$. For example, the sensitivities of the upper bound on the consistency measure with respect

to an upper parameter bound and an upper experiment bound are

$$\begin{aligned}\left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial \log_{10}(\overline{k}_i)}\right|_{\substack{\underline{k}_i=\hat{\underline{k}}_i \\ \overline{k}_i=\hat{\overline{k}}_i}} &= \left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial \beta_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} \cdot \left.\frac{\partial \phi_i}{\partial \log_{10}(k_i)}\right|_{\substack{\underline{k}_i=\hat{\underline{k}}_i \\ \overline{k}_i=\hat{\overline{k}}_i}} \\ &= \frac{\nu^{\star}_{i,\beta}(Q^{\star}_{(1,i+1)} - \hat{\alpha}_i + \epsilon_i)}{\log_{10}(s_i)},\end{aligned} \tag{4.50}$$

$$\begin{aligned}\left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial \log_{10}(\mathfrak{u}_e)}\right|_{\mathfrak{u}_e=\hat{\mathfrak{u}}_e} &= \left.\frac{\partial \overline{\mathrm{C}}_{\mathcal{D}}}{\partial u_e}\right|_{u_e=\hat{u}_e} \\ &= \lambda^{\star}_{e,u}\left(1 - \overline{\mathrm{C}}_{\mathcal{D}}(u_e = \hat{u}_e)\right).\end{aligned} \tag{4.51}$$

The other sensitivities have similar variable transformations.

Figure 4.2 shows the sensitivities of the upper bound on the consistency measure with respect to each of the bounds on the 77 experiments and 102 parameters. Locally the upper bound on the consistency measure is most sensitive to the upper bound on experiment 36 (target F4 [28]) and experiment 37 (target F5 [53]). To examine these sensitivities further, we remove each experiment constraint from the dataset, one at a time, and recalculate the consistency measure. When target F4 is removed, the consistency measure bounds were calculated as $[-0.062, 0.021]$. These bounds do not ascertain the consistency of the dataset; however, after one branch and bound iteration the consistency measure was tightened to $[-0.062, -0.009]$, showing inconsistency. When we instead remove target F5 (leaving target F4 in the dataset), the consistency bounds are calculated as [0.13, 0.24]. Figure 4.3 shows the sensitivities of the upper bound on the consistency measure with target F5 removed. After removing target F5, the scale of the sensitivities is reduced and spread out among more experiments, and it is still relatively insensitive to the parameters.

Since the GRI-Mech 3.0 dataset with the new experiment uncertainties is inconsistent with target F5 included, we shall remove target F5 from the dataset for any examples throughout the rest of this document. Target F5 would now seem a good candidate for a response prediction, as it is no longer in the dataset; however, this prediction is presented in [121] and doesn't exhibit very interesting sensitivities as an example. We therefore, remove target StF8, the laminar flame speed in a stoichiometric atmospheric ethane-air mixture [113], and predict its range. Predicting target StF8 using the rest of the GRI-Mech 3.0 dataset (except target F5) yields an outer bound interval of [1.545, 1.727] $\log_{10}$(cm/s) and an inner bound interval of [1.566 1.699] $\log_{10}$(cm/s).

Figures 4.4 and 4.5 show the sensitivities related to the outer bounds $\underline{L}_0$ and $\overline{R}_0$, respectively. These sensitivities provide directions for improving the prediction of target StF8 (i.e. narrowing the range of the prediction) by demonstrating which experiment or parameter uncertainties have the most effect on the prediction bounds. Figure 4.4 shows the left outer prediction bound is most sensitive to the lower uncertainty bound on the 37^{th} target (target F6 [53]). Figure 4.5 shows the right outer prediction bound is most sensitive to the upper uncertainty bound on the 67^{th} target (target SF7 [68]).

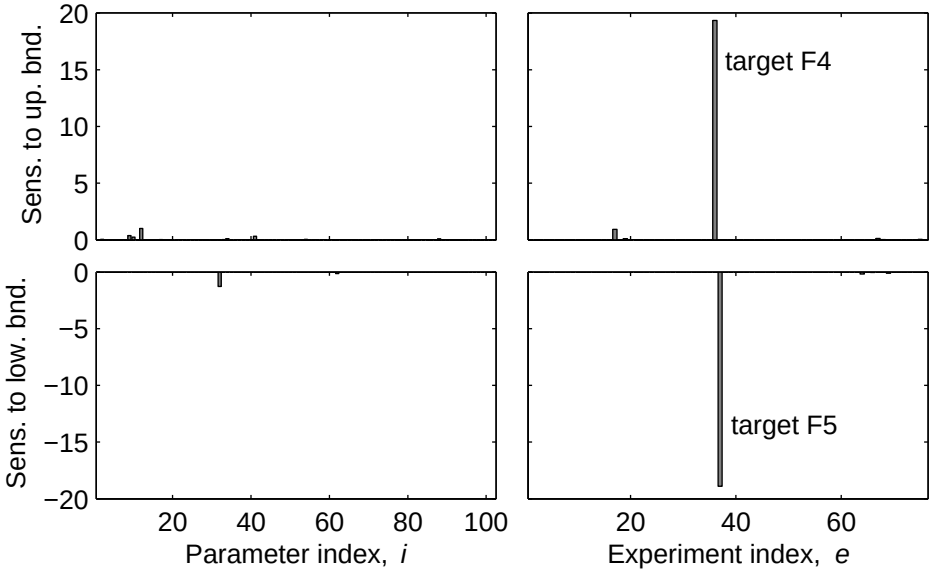

Figure 4.2: Sensitivity to the upper bound on the consistency measure $\overline{\mathrm{C}}_{\mathcal{D}}$ of the GRI-Mech 3.0 dataset with respect to the bounds on its 102 parameters and 77 experiments. The inconsistent measure has the most dependence on two experiments, target F4 [28] and target F5 [53].

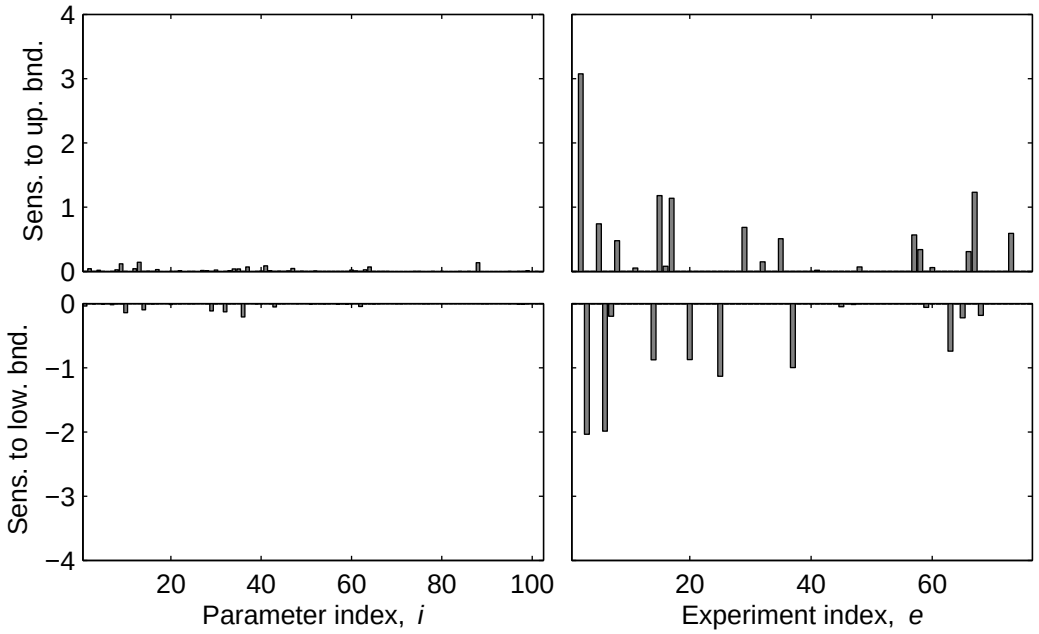

Figure 4.3: Sensitivity to the upper bound on the consistency measure $\overline{\mathrm{C}}_{\mathcal{D}}$ of the GRI-Mech 3.0 dataset with respect to the bounds on its 102 parameters and 76 of its experiments. Target F5 [53] is not included in the dataset. The dataset is consistent, and the consistency measure is not overly sensitive to any one or two experiments or parameters.

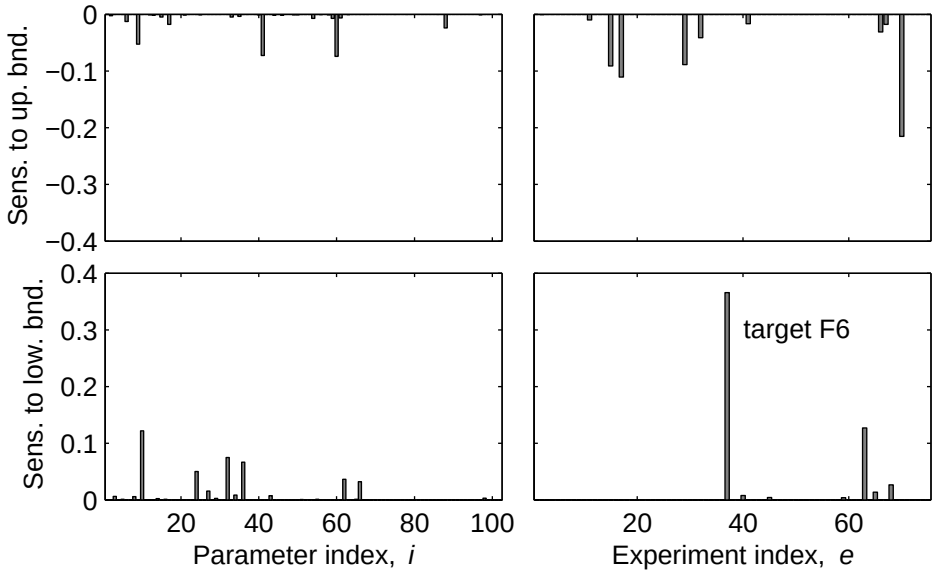

Figure 4.4: Sensitivity of the lower (outer) bound on the left endpoint, $\underline{L}_0$, of the prediction of target StF8 using the constraints of the rest of the GRI-Mech 3.0 dataset with respect to the lower and upper bounds on the 102 parameters and 76 experiments.

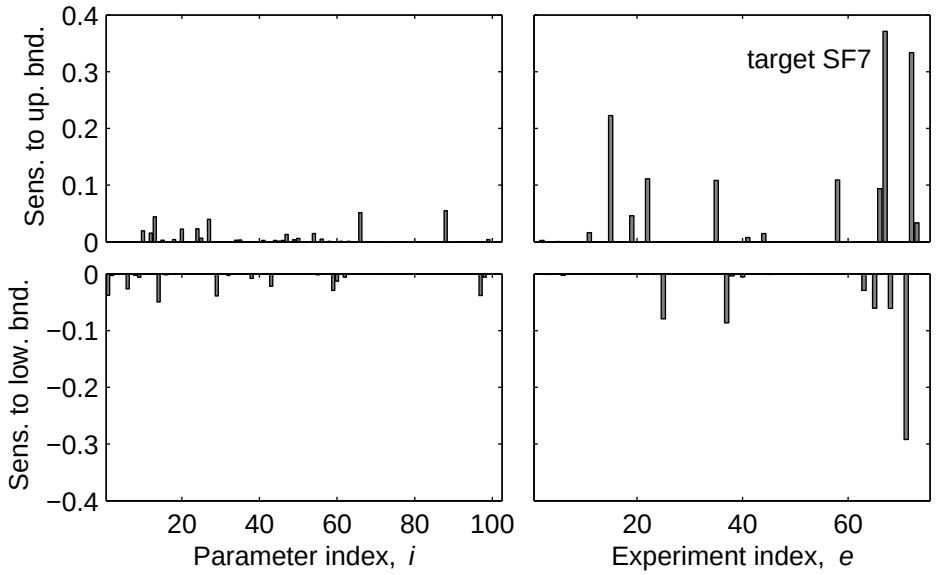

Figure 4.5: Sensitivity of the upper (outer) bound on the right endpoint, $\overline{R}_0$, of the prediction of target StF8 using the constraints of the rest of the GRI-Mech 3.0 dataset with respect to the lower and upper bounds on the 102 parameters and 76 experiments.

The sensitivity of the left outer prediction endpoint with respect to the upper bound uncertainty in target F6 is about 0.366. The lower bound uncertainty for F6 (as reported by Xiaoqing You [120, 121]) is -0.0448. Hypothetically, if further experiments increased this lower uncertainty by 0.02 to -0.0248, we would expect the left outer prediction endpoint to increase by at least $0.366 \cdot 0.02 = 0.0073$ to 1.552. We changed the lower uncertainty of target F6 to -0.0248 and re-performed the prediction of StF8. The new outer left endpoint is 1.553, within 0.01% of the estimate.

4.6 Assessing a General Experiment and Parameter Impact

When gauging the importance of the uncertainty in a parameter or experiment relative to a response prediction, the sensitivities derived in Section 4.3.1 are for the prediction of a specific observable. A more general measure of parameter and experiment impact could be useful for helping modelers and experimenters concentrate their efforts. Specifically, we would like a prediction impact measure that is more independent of the observable being predicted.

First we derive the sensitivity of the prediction interval length to the lengths of the experiment/parameter uncertainty intervals. Take for example the length of the experiment uncertainty interval in experiment e, $unc_e = u_e - l_e$. We make the assumption that a tightening of unc_e is symmetric, therefore for a given prediction endpoint, the sensitivities with respect to u_e and l_e are averaged to obtain a sensitivity with respect to unc_e. Based on this assumption, the sensitivity of the prediction interval length $\overline{I}_0 = \overline{R}_0 - \underline{L}_0$ with respect to unc_e is

$$\frac{\partial \overline{I}_0}{\partial unc_e} = \frac{1}{2}\left(\frac{\partial \overline{R}_0}{\partial u_e} - \frac{\partial \overline{R}_0}{\partial l_e} - \frac{\partial \underline{L}_0}{\partial u_e} + \frac{\partial \underline{L}_0}{\partial l_e}\right). \tag{4.52}$$

A similar expression is derived for the sensitivity due to the uncertainty interval length in a parameter x_i as

$$\frac{\partial \overline{I}_0}{\partial unc_i} = \frac{1}{2}\left(\frac{\partial \overline{R}_0}{\partial \beta_i} - \frac{\partial \overline{R}_0}{\partial \alpha_i} - \frac{\partial \underline{L}_0}{\partial \beta_i} + \frac{\partial \underline{L}_0}{\partial \alpha_i}\right). \tag{4.53}$$

These expressions provide a single sensitivity for the uncertainty in each parameter and in each experiment.

To assess a general impact factor that each experiment and parameter has on the model prediction, we need a large collection of models of observables to predict, $\{M_{0,j}\}_{j=1}^{N}$. This collection should include a representative sample of models in the dataset's domain. Predictions are then performed on each model $M_{0,j}$ and for each experiment and parameter. The associated sensitivities are then averaged over all predictions. These average sensitivities are our general measures of sensitivity [95].

In lieu of an available collection, we pose two possible alternatives. First, predict the range of each model in the dataset, M_e, constrained to the rest of the dataset experiments. If the dataset itself has a fair number of models, this could provide enough predictions to average out the effects of model specific prediction. It also trivially covers the type of observables present in the dataset. The top row of Figure 4.6 shows these averaged sensitivities for the GRI-Mech 3.0 dataset.

The second method is to generate a set of models from a random distribution. For each model to be generated, we first pick a list of parameters. This list is chosen such that each pair of parameters in the list both appear as active parameters in at least one of the dataset models. Each of the dataset models has been fitted with a quadratic surrogate model $M_e(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{A}_e\mathbf{x} + \mathbf{b}_e^\mathsf{T}\mathbf{x} + c_e$. The random model is also quadratic, and its coefficients are constrained to be the same average scale as those in the dataset models. Specifically,

$$\|\mathbf{A}_{0,j}\|_\mathrm{F} = \sum_{e=1}^{m} \|\mathbf{A}_e\|_\mathrm{F}, \quad \|\mathbf{b}_{0,j}\|_\mathrm{F} = \sum_{e=1}^{m} \|\mathbf{b}_e\|_\mathrm{F}, \tag{4.54}$$

where $\|\cdot\|_\mathrm{F}$ is the Frobenius norm. The constant term $c_{0,j}$ is arbitrarily set to 0 as it does not affect the prediction sensitivities. These random models, should be similar to the dataset models and there are an arbitrary number of them. The bottom row of Figure 4.6 shows the average sensitivities over 10,000 random models of this type with respect to the GRI-Mech 3.0 dataset.

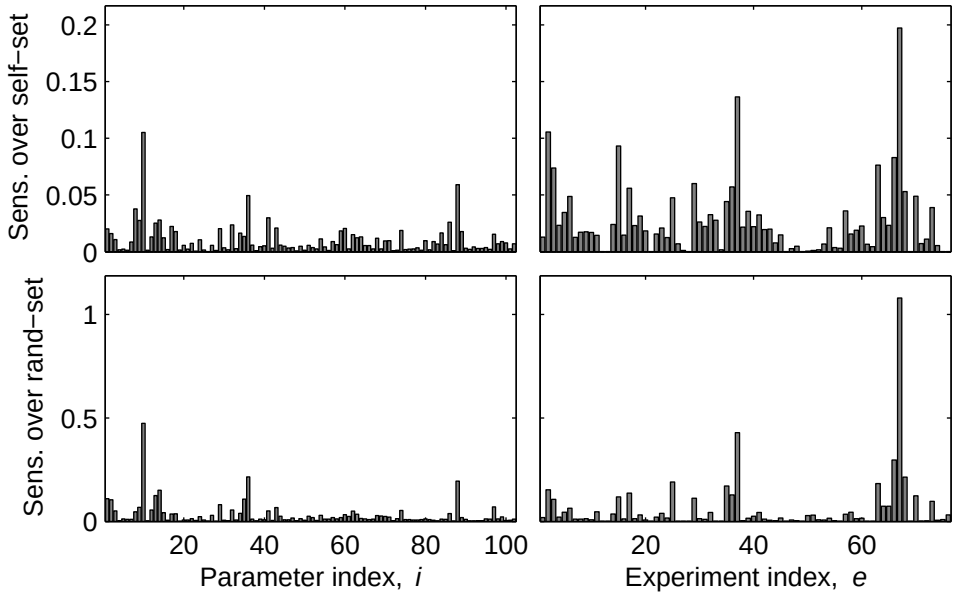

Figure 4.6: Average sensitivities over a collection of prediction models with respect to the GRI-Mech 3.0 parameter and experiment uncertainties. The top row shows the average over predictions of each of the GRI-Mech models constrained by the rest of the dataset. The bottom row shows the average over predictions of random quadratic models generated in the manner described in the text.

The sensitivities in Figure 4.6 show similarities between the two methods of averaging over GRI-Mech 3.0 models and random models. Both techniques share some of the larger sensitivities (although on a different scale); however, the averaging over the 10,000 random models exhibits more sparsity in the sensitivities.

4.7 Prediction Linearization Using Sensitivity Information

With sensitivity information, a linearization of the response prediction as a function of experiment and parameter uncertainty is easy to create. Specifically, first order Taylor series approximations of the left and right prediction endpoints only require an evaluation of the prediction and the first derivatives (i.e. the sensitivities). Explicitly,

$$\begin{aligned}
\underline{L}_0(\mathbf{l},\mathbf{u},\boldsymbol{\alpha},\boldsymbol{\beta}) \approx & \underline{L}_0(\hat{\mathbf{l}},\hat{\mathbf{u}},\hat{\boldsymbol{\alpha}},\hat{\boldsymbol{\beta}}) + \dots \\
& + \sum_{e=1}^{m} \left[(l_e - \hat{l}_e) \cdot \left.\frac{\partial \underline{L}_0}{\partial l_e}\right|_{l_e=\hat{l}_e} + (u_e - \hat{u}_e) \cdot \left.\frac{\partial \underline{L}_0}{\partial u_e}\right|_{u_e=\hat{u}_e} \right] + \dots \\
& + \sum_{i=1}^{n} \left[(\alpha_i - \hat{\alpha}_i) \cdot \left.\frac{\partial \underline{L}_0}{\partial \alpha_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} + (\beta_i - \hat{\beta}_i) \cdot \left.\frac{\partial \underline{L}_0}{\partial \beta_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} \right]
\end{aligned} \tag{4.55}$$

$$\begin{aligned}
\overline{R}_0(\mathbf{l},\mathbf{u},\boldsymbol{\alpha},\boldsymbol{\beta}) \approx & \overline{R}_0(\hat{\mathbf{l}},\hat{\mathbf{u}},\hat{\boldsymbol{\alpha}},\hat{\boldsymbol{\beta}}) + \dots \\
& + \sum_{e=1}^{m} \left[(l_e - \hat{l}_e) \cdot \left.\frac{\partial \overline{R}_0}{\partial l_e}\right|_{l_e=\hat{l}_e} + (u_e - \hat{u}_e) \cdot \left.\frac{\partial \overline{R}_0}{\partial u_e}\right|_{u_e=\hat{u}_e} \right] + \dots \\
& + \sum_{i=1}^{n} \left[(\alpha_i - \hat{\alpha}_i) \cdot \left.\frac{\partial \overline{R}_0}{\partial \alpha_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} + (\beta_i - \hat{\beta}_i) \cdot \left.\frac{\partial \overline{R}_0}{\partial \beta_i}\right|_{\substack{\alpha_i=\hat{\alpha}_i \\ \beta_i=\hat{\beta}_i}} \right]
\end{aligned} \tag{4.56}$$

where the partial derivatives are those derived in equations (4.18) and (4.22).

Prediction linearization has many potential applications. If the prediction is to be constrained or optimized in any way, the prediction will possibly be evaluated many times with different uncertainty values by an optimization solver. If each prediction takes several minutes (or hours) to compute, this could be very computationally expensive. Approximating the prediction with a linearization with alleviate much of that expense.

One example where linearization is beneficial is a cost-constrained uncertainty quantification. The sensitivities by themselves provide insight into possible future experiments that would best improve the predictive quality of the dataset. However, different experiments and data collection naturally would each have their own real-

world costs; it might be more costly to reduce the uncertainty in the experiment associated with the highest sensitivity than to do so with a lower ranked experiment.

Let $K_{e,l}$ be the cost associated with the lower uncertainty bound from the e^{th} constraint. As a function, $K_{e,l}(l_e)$ outputs the cost of tightening this uncertainty from the current value of $\hat{l}_e$ to l_e. The cost function should exhibit some practical behaviors; the cost is zero at $\hat{l}_e$, it increases in value as the uncertainty tightens, and it becomes infinite or at least very large with no uncertainty. Similar costs are defined for the experiment upper uncertainty and the lower and upper parameter uncertainties, i.e. $K_{e,u}$, $K_{i,\alpha}$, and $K_{i,\beta}$. For notational simplicity, we define a $2n \times 2m$ vector $\boldsymbol{\delta}$ that contains all the uncertainty values, and $K_j(\delta_j)$ is the cost for the j^{th} element of $\boldsymbol{\delta}$.

One generic goal is to reduce the length of the prediction interval in a cost effective manner. Since prediction interval length reduction and cost reduction are two competing objectives, one is minimized while the other is used as a constraint. Given a budget T, the uncertainty that achieves the minimal prediction interval length within budget is written as,

$$\begin{aligned} \boldsymbol{\delta}^{\star}_{\text{min.pred.}} = &\underset{\boldsymbol{\delta}}{\operatorname{argmin}}\, \overline{R}_0(\boldsymbol{\delta}) - \underline{L}_0(\boldsymbol{\delta}) \\ &\text{s.t.} \left\{ \begin{array}{l} \sum_{j=1}^{2n+2m} K_j(\delta_j) \leq T, \\ 0 \leq \delta_j \leq \hat{\delta}_j,\ j = 1, \ldots, 2n+2m. \end{array} \right\} \end{aligned} \tag{4.57}$$

Given a desired prediction interval length I, the uncertainty that achieves this prediction length with the minimal cost is written as,

$$\begin{aligned} \boldsymbol{\delta}^{\star}_{\text{min.cost}} = &\underset{\boldsymbol{\delta}}{\operatorname{argmin}} \sum_{j=1}^{2n+2m} K_j(\delta_j) \\ &\text{s.t.} \left\{ \begin{array}{l} \overline{R}_0(\boldsymbol{\delta}) - \underline{L}_0(\boldsymbol{\delta}) \leq I \\ 0 \leq \delta_j \leq \hat{\delta}_j,\ j = 1, \ldots, 2n+2m. \end{array} \right\} \end{aligned} \tag{4.58}$$

Both $\boldsymbol{\delta}^{\star}_{\text{min.pred.}}$ and $\boldsymbol{\delta}^{\star}_{\text{min.cost}}$ are difficult to calculate because $\underline{L}_0(\boldsymbol{\delta})$ and $\overline{R}_0(\boldsymbol{\delta})$ are optimization problems themselves, potentially taking several minutes per evaluation. This is where the application of the prediction linearization is useful. With the linearization, the optimizations in equations (4.57) and (4.58) are approximated

respectively as,

$$\begin{aligned}
\boldsymbol{\delta}^{\star}_{\text{min.pred.}} &\approx \underset{\boldsymbol{\delta}}{\operatorname{argmin}}\, \overline{R}_0(\hat{\boldsymbol{\delta}}) - \underline{L}_0(\hat{\boldsymbol{\delta}}) + \sum_{j=1}^{2n+2m} \left(\delta_j - \hat{\delta}_j\right) \cdot \left(\left.\frac{\partial \overline{R}_0}{\partial \delta_j}\right|_{\boldsymbol{\delta}=\hat{\boldsymbol{\delta}}} - \left.\frac{\partial \underline{L}_0}{\partial \delta_j}\right|_{\boldsymbol{\delta}=\hat{\boldsymbol{\delta}}} \right) \\
&\qquad \text{s.t.} \begin{cases} \sum_{j=1}^{2n+2m} K_j(\delta_j) \le T, \\ 0 \le \delta_j \le \hat{\delta}_j,\ j = 1, \ldots, 2n+2m, \end{cases}
\end{aligned} \tag{4.59}$$

$$\begin{aligned}
\boldsymbol{\delta}^{\star}_{\text{min.cost}} &\approx \underset{\boldsymbol{\delta}}{\operatorname{argmin}} \sum_{j=1}^{2n+2m} K_j(\delta_j) \\
&\qquad \text{s.t.} \begin{cases} \overline{R}_0(\hat{\boldsymbol{\delta}}) - \underline{L}_0(\hat{\boldsymbol{\delta}}) + \sum_{j=1}^{2n+2m} \left(\delta_j - \hat{\delta}_j\right) \cdot \left(\left.\frac{\partial \overline{R}_0}{\partial \delta_j}\right|_{\boldsymbol{\delta}=\hat{\boldsymbol{\delta}}} - \left.\frac{\partial \underline{L}_0}{\partial \delta_j}\right|_{\boldsymbol{\delta}=\hat{\boldsymbol{\delta}}} \right) \le I \\ 0 \le \delta_j \le \hat{\delta}_j,\ j = 1, \ldots, 2n+2m. \end{cases}
\end{aligned} \tag{4.60}$$

If the cost functions K_j are convex, then these are convex optimizations and efficient to solve with readily available solvers.

4.7.1 Example: Cost Analysis of GRI-Mech 3.0 Prediction Using Linearization

The prediction of target StF8 using the rest of the GRI-Mech 3.0 dataset produces the sensitivity information in Figures 4.4 and 4.5 in Section 4.5. The prediction was most sensitive to the lower uncertainty of target F6 and the upper uncertainty of target SF7. To reduce computation complexity we temporarily only focus on those two uncertainties.

We assign to each uncertainty δ a cost function which has the desired properties,

$$K(\delta) = \left(\frac{\hat{\delta}}{\delta}\right)^{a} - 1. \tag{4.61}$$

where a is a positive number that relates to how quickly the cost rises with the uncertainty reduction. When the uncertainty is held at the "current" uncertainty $\hat{\delta}$, the cost is 0. Elimination of all uncertainty results in an infinite cost. Furthermore, K is a convex function. For the present example, the parameters are arbitrarily chosen as $a_{\text{F4}} = 3$ and $a_{\text{SF7}} = 2$.

As already mentioned, solving the full non-linearized cost minimization problem (4.58) is difficult and we therefore allow only the lower uncertainty of target F6 and the upper uncertainty of target SF7 to vary. The budget is set to $T = 0.3$. Solving the two variable non-linearized problem obtains the optimal uncertainties of $l^{\star}_{F4} = -0.0401$ and $u^{\star}_{SF7} = 0.0042$, which is no change in l_{F4} and a 0.0006 decrease in

u_{SF7}. This calculation took about 2 hours and 28 minutes to compute. Solving the two variable linearized problem takes about 0.35 seconds to compute (plus about 2.5 minutes for the initial prediction to get sensitivities). and obtains the same answer to 6 digits of precision. An optimization over all 354 uncertainties (2 for each of the 102 parameters and 2 for each the 75 experiments) with the linearized prediction took about 2.5 minutes (again plus another 2.5 mintues for the initial prediction).

Chapter 5

Representative Feasible Parameter Vector

Every point in the feasible set $\mathcal{F}$ is a vector of viable parameter values based on the deterministic view of uncertainty presented in Chapter 2. However, there is often a value associated with each parameter that is set by the scientific community to be designated the "nominal value" of the parameter. This representative point can be useful in designing experiments and running simulations.

The nominal value is sometimes determined by experiments on a parameter-by-parameter basis, which often places the nominal value in the center of its uncertainty range. Sometimes the nominal values are chosen via parameter optimization techniques [39, 44, 52, 104]. Both of these techniques can result in an infeasible choice of nominal values. This chapter presents one way of choosing a nominal parameter vector that is feasible. It is an expansion on some of the work presented in [121].

5.1 Minimizing the Number of Parameter Deviations From Nominal Values

The literature value of each parameter is usually presented as the result of some combination of careful experimentation, tested analysis techniques, and expert opinion. If a collection of experiments and models invalidate the nominal parameter vector (i.e. it is not in the feasible set), then clearly the parameter values must deviate from the nominal values in order to achieve feasibility. However, due to the amount of work

that went into the determination of these nominal values, experts are often reluctant to accept such deviations.

This presents a natural question: what is the minimal number of parameter deviations away from the nominal values required to achieve feasibility? We pose this question mathematically as an optimization problem using the cardinality function, $\text{card}(\cdot)$.

$$\text{MD}^{\star}_{\text{card}} := \underset{\mathbf{x}\in\mathcal{F}}{\text{argmin}}\, \text{card}\,(\mathbf{x}-\mathbf{x}_0) \tag{5.1}$$

where $\mathbf{x}_0$ is the nominal parameter vector. The cardinality function counts the number of nonzero elements of its argument and is sometimes called the ℓ^0- norm (although it is not a true norm).[1] This optimization problem is an instance of a Minimal Cardinality Program (MCP).

One possible way to solve this optimization problem is to freeze a set of parameters at the nominal values and solve the feasibility problem. In the worst case we must try every possible combination of parameters to freeze in order to find the minimum number. Unfortunately, there are 2^n possible combinations for n parameters and the problem is NP-hard [82]. The next section discusses a relaxation for the cardinality problem in equation (5.1) using the ℓ^1-norm.

5.2 Convex Approximation Using the One-Norm

For a convex domain $\mathcal{C}$, the convex envelope of a function f is the largest convex function g such that $g(\mathbf{x}) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{C}$. In other words, the convex envelope is the best pointwise, convex, lower-bound approximation on the domain $\mathcal{C}$ [89]. The convex envelope of the rank function on the set $\{\mathbf{X} \in \mathbb{R}^{m\times n} : \overline{\sigma}(\mathbf{X}) \leq 1\}$ is the nuclear norm (the sum of the eigenvalues) [31, 89]. It can be easily shown that this implies that the convex envelope of the cardinality function on the set $\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_\infty \leq 1\}$ is the ℓ^1-norm (the sum of the absolute values of the vector elements) [31]. Using the ℓ^1-norm in minimization is a well-known heuristic approximation to the cardinality function [17, 24] as it tends to drive the elements of its input to zero.

Applying this ℓ^1-norm approximation heuristic, the minimal deviation MCP (equation (5.1)) is approximated as,

$$\text{MD}^{\star}_{\ell_1} := \min_{\mathbf{x}\in\mathcal{F}} \|\mathbf{x}-\mathbf{x}_0\|_1 = \min_{\mathbf{x}\in\mathcal{F}} \sum_{i=1}^{n} |x_i|. \tag{5.2}$$

If a normalization of the parameters is used, the prior bounds hyperrectangle is constrained to be $\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_\infty \leq 1\}$. In this case, the ℓ^1-norm is the convex

[1] The ℓ^p-norm is defined as $\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p\right)^{1/p}$ for $p \geq 1$. The triangle inequality does not hold for $0 < p < 1$ and hence $\|\mathbf{x}\|_p$ is not a norm for these values of p. The cardinality function is the limit as p goes to 0: $\text{card}(\mathbf{x}) = \lim_{p\to 0} \|\mathbf{x}\|_p$

envelope, and hence a lower bound, of the cardinality function. Since the feasible $\mathcal{F}$ is a subset of $\mathcal{H}$, $\mathrm{MD}^\star_{\ell_1} \leq \mathrm{MD}^\star_{\mathrm{card}}$. Note that $\mathrm{MD}^\star_{\ell_1} = \mathrm{MD}^\star_{\mathrm{card}}$ if $\mathbf{x}_0 \in \mathcal{F}$. Therefore, for the rest of this chapter, we assume that the parameters have been normalized.

5.3 NQCQP Formulation of ℓ^1-Approximation

The absolute value function, and by extension the ℓ^1-norm, can be rewritten as a linear function of twice as many variables with linear constraints. This is achieved by separating the positive and negative elements of the vector $\mathbf{x} - \mathbf{x}_0$ [36]. Define two positive vectors $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$, such that $\mu_i \nu_i = 0$ (i.e. for each index i, only one of the vectors has a nonzero entry) and $x_i - x_{0,i} = \mu_i - \nu_i$. In other words, μ_i is equal to $x_i - x_{0,i}$ when $x_i - x_{0,i}$ is positive and is zero otherwise, whereas ν_i is equal to $x_{0,i} - x_i$ when it is positive and is zero otherwise. Then $|x_i - x_{0,i}| = \mu_i + \nu_i$.

The ℓ^1-norm minimization problem in equation (5.2) can be rewritten using the vectors $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$.

$$\begin{aligned} \mathrm{MD}^\star_{\ell_1} = & \min_{\boldsymbol{\mu},\boldsymbol{\nu}} \sum_{i=1}^{n} \mu_i + \nu_i \\ & \text{s.t.} \begin{cases} \boldsymbol{\mu} - \boldsymbol{\nu} + \mathbf{x}_0 \in \mathcal{F} \\ \boldsymbol{\mu} \geq 0,\, \boldsymbol{\nu} \geq 0 \\ \mu_i \nu_i = 0,\, i = 1, \ldots, n \end{cases} \end{aligned} \tag{5.3}$$

If all of the constraint functions that make up the feasible set $\mathcal{F}$ are quadratic, then this formulation is an NQCQP, and the techniques described in Chapter 3 can be utilized. The next few sections discuss the explicit formulation of equation (5.3) as the NQCQP found in Chapter 3.

5.3.1 Combining Positive Constraints and Box Constraints

The prior information (box) constraints confine each parameter to an interval.

$$\alpha_i \leq x_i \leq \beta_i \tag{5.4}$$

Substituting the vectors $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ used in equation (5.3) yields,

$$\alpha_i \leq \mu_i - \nu_i + x_{0,i} \leq \beta_i. \tag{5.5}$$

The equality constraint $\mu_i \nu_i = 0$ forces either μ_i or ν_i to be zero. Therefore we can split the box constraint up.

$$\begin{aligned} \alpha_i - x_{0,i} \leq \mu_i \leq \beta_i - x_{0,i} \\ \alpha_i - x_{0,i} \leq -\nu_i \leq \beta_i - x_{0,i} \end{aligned} \tag{5.6}$$

We assume the nominal value is in the prior bounds and add the positive constraint for μ_i and ν_i.

$$\begin{aligned} 0 \leq \mu_i \leq \beta_i - x_{0,i} \\ 0 \leq \nu_i \leq x_{0,i} - \alpha_i \end{aligned} \tag{5.7}$$

We now have a lower and upper bound for each element of $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$. In Section 3.4, the upper and lower bounds on the parameters were formulated as two quadratic constraints to obtain Lagrange multipliers for each constraint. Applying this same technique using a small $\epsilon > 0$ (or a different one for each constraint) gives four quadratic constraints.

$$\begin{aligned} \mu_i(\mu_i - \beta_i + x_{i,0} - \epsilon) \leq 0 \\ (\mu_i + \epsilon)(\mu_i - \beta_i + x_{i,0}) \leq 0 \\ \nu_i(\nu_i - x_{i,0} + \alpha_i - \epsilon) \leq 0 \\ (\nu_i + \epsilon)(\nu_i - x_{i,0} + \alpha_i) \leq 0 \end{aligned} \tag{5.8}$$

Let the $\boldsymbol{\omega}$ denote the vector of $\boldsymbol{\mu}$ stacked over $\boldsymbol{\nu}$, i.e.

$$\boldsymbol{\omega} := \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\nu} \end{bmatrix} \tag{5.9}$$

Then each of these $4n$ quadratic constraints on the vectors $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ can be written in the matrix form as

$$\left[\begin{smallmatrix} 1 \\ \boldsymbol{\omega} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{P}_i \left[\begin{smallmatrix} 1 \\ \boldsymbol{\omega} \end{smallmatrix}\right] \leq 0, \tag{5.10}$$

where $\mathbf{P}_i$ is a $4n + 1 \times 4n + 1$ matrix representing the quadratic coefficients.

5.3.2 Quadratic Experiment Constraint Formulation

Other than the box constraints on the parameters, there are the constraints due to models and experiments to consider. Using the notation from Chapter 3 each of these quadratic constraints can be written as

$$\left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right] \leq 0. \tag{5.11}$$

Substituting the $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ vectors used in equation (5.3) we get,

$$\left[\begin{smallmatrix} 1 \\ \boldsymbol{\mu} - \boldsymbol{\nu} + \mathbf{x}_0 \end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix} 1 \\ \boldsymbol{\mu} - \boldsymbol{\nu} + \mathbf{x}_0 \end{smallmatrix}\right] \leq 0. \tag{5.12}$$

Define the matrix $\mathbf{P}_i$ as,

$$\mathbf{P}_i := \begin{bmatrix} 1 & x_0^{\mathsf{T}} \\ \mathbf{0} & I \\ \mathbf{0} & -I \end{bmatrix} \mathbf{Z}_i \begin{bmatrix} 1 & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_0 & I & -I \end{bmatrix} \tag{5.13}$$

Then using the vector $\boldsymbol{\omega}$ as defined in equation (5.9), the constraint in equation (5.11) is rewritten as,

$$\left[\begin{smallmatrix}1\\ \boldsymbol{\omega}\end{smallmatrix}\right]^{\mathsf{T}} \mathbf{P}_i \left[\begin{smallmatrix}1\\ \boldsymbol{\omega}\end{smallmatrix}\right] \leq 0 \tag{5.14}$$

which is a quadratic constraint in $2n$ variables. Combined with the box constraints derived in the previous section we re-index the matrices and let i range from 1 to $4n + 2m$ for the $4n$ box constraints and the $2m$ experiment constraints.

5.3.3 Equality Constraint Formulation

For each index $j = 1, \ldots, n$, either μ_j or ν_j needs to be 0. There are several ways we could enforce this requirement as a quadratic constraint.

Method 1: As shown in the formulation shown in equation (5.3), the constraint can be enforced with a quadratic constraint for each index $j = 1, \ldots, n$.

$$\mu_j \nu_j = 0 \tag{5.15}$$

Each of these quadratic constraints can be written in matrix form. Define the matrix $\mathcal{P}_j$ as

$$\mathcal{P}_j := \begin{bmatrix} 0 & \mathbf{0}^{\mathsf{T}} & \mathbf{0}^{\mathsf{T}} \\ \mathbf{0} & \mathbf{0}_n & \mathbf{B}_{jj} \\ \mathbf{0} & \mathbf{B}_{jj} & \mathbf{0}_n \end{bmatrix} \tag{5.16}$$

where $\mathbf{B}_{jj}$ is an $n \times n$ standard basis matrix, i.e. a matrix with a one at row j column j and zeros elsewhere. The equality constraints are then written as

$$\left[\begin{smallmatrix}1\\ \boldsymbol{\omega}\end{smallmatrix}\right]^{\mathsf{T}} \mathcal{P}_j \left[\begin{smallmatrix}1\\ \boldsymbol{\omega}\end{smallmatrix}\right] = 0, \; j = 1, \ldots, n. \tag{5.17}$$

Method 2: Since $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are constrained to be positive, the equality constraints can be written as one quadratic constraint.

$$\boldsymbol{\mu}^{\mathsf{T}} \boldsymbol{\nu} = 0 \tag{5.18}$$

Define the matrix $\mathcal{P}$ as

$$\mathcal{P} := \begin{bmatrix} 0 & \mathbf{0}^{\mathsf{T}} & \mathbf{0}^{\mathsf{T}} \\ \mathbf{0} & \mathbf{0}_n & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{0}_n \end{bmatrix}. \tag{5.19}$$

Then this equality constraint is written as

$$\left[\begin{smallmatrix}1\\ \boldsymbol{\omega}\end{smallmatrix}\right]^{\mathsf{T}} \mathcal{P} \left[\begin{smallmatrix}1\\ \boldsymbol{\omega}\end{smallmatrix}\right] = 0. \tag{5.20}$$

Method 3: The constraint is not explicitly needed, because the optimal $\boldsymbol{\mu}^\star$ and $\boldsymbol{\nu}^\star$ will satisfy $\mu_i^\star \nu_i^\star = 0$ for each i.

Proof. We prove this by assuming the optimal variables don't satisfy the constraint, and showing a contradiction. Suppose there exists a $j \in \{1, 2, \ldots, n\}$ such that $\mu_j^\star \nu_j^\star \neq 0$. For each $i = 1, \ldots, n$ let $\epsilon_i := \min\{\mu_i^\star, \nu_i^\star\}$. Each $\epsilon_i \geq 0$, and $\epsilon_j > 0$ since $\mu_j^\star \nu_j^\star \neq 0$. Let $\tilde{\boldsymbol{\mu}} = \boldsymbol{\mu}^\star - \boldsymbol{\epsilon}$ and $\tilde{\boldsymbol{\nu}} = \boldsymbol{\nu}^\star - \boldsymbol{\epsilon}$. The vectors $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\nu}}$ are feasible points: $\tilde{\boldsymbol{\mu}} - \tilde{\boldsymbol{\nu}} + \mathbf{x}_0 = \boldsymbol{\mu}^\star - \boldsymbol{\nu}^\star + \mathbf{x}_0 \in \mathcal{F}$, $\tilde{\boldsymbol{\mu}} \geq 0$, and $\tilde{\boldsymbol{\nu}} \geq 0$. The cost using the vectors $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\nu}}$ is less than the cost with vectors $\boldsymbol{\mu}^\star$ and $\boldsymbol{\nu}^\star$.

$$\sum_{i=1}^{n} \tilde{\mu}_i + \tilde{\nu}_i = \sum_{i=1}^{n} \mu_i^\star + \nu_i^\star - 2\epsilon_i < \sum_{i=1}^{n} \mu_i^\star + \nu_i^\star$$

The inequality is strict because $\epsilon_j > 0$. Therefore $\boldsymbol{\mu}^\star$ and $\boldsymbol{\nu}^\star$ are not optimal. Contradiction. □

All of these constraints are equivalent in the primal form of the NQCQP. However, the choice can affect the lower bound formulation using the rank relaxation (or S-procedure). We now formulate three lower bounds on the optimum using the different equality constraints, and discuss their difference.

Letting $\mathbf{P}_0$ denote the quadratic matrix associated with the objective function, the lower bound using the rank relaxation as discussed in Section 3.1.2 can now be formulated using each of the three equality constraint cases, via,

$$\underline{p}_1^\star := \min_{\mathbf{Q} \succeq 0,\, Q_{11}=1} \operatorname{Tr}[\mathbf{P}_0 \mathbf{Q}] \quad \text{s.t.} \begin{cases} \operatorname{Tr}[\mathbf{P}_i \mathbf{Q}] \leq 0,\ i = 1, \ldots, 4n + 2m, \\ \operatorname{Tr}[\mathcal{P}_j \mathbf{Q}] = 0,\ j = 1, \ldots, n, \end{cases} \tag{5.21}$$

$$\underline{p}_2^\star := \min_{\mathbf{Q} \succeq 0,\, Q_{11}=1} \operatorname{Tr}[\mathbf{P}_0 \mathbf{Q}] \quad \text{s.t.} \begin{cases} \operatorname{Tr}[\mathbf{P}_i \mathbf{Q}] \leq 0,\ i = 1, \ldots, 4n + 2m, \\ \operatorname{Tr}[\mathcal{P} \mathbf{Q}] = 0, \end{cases} \tag{5.22}$$

$$\underline{p}_3^\star := \min_{\mathbf{Q} \succeq 0,\, Q_{11}=1} \operatorname{Tr}[\mathbf{P}_0 \mathbf{Q}] \quad \text{s.t. } \operatorname{Tr}[\mathbf{P}_i \mathbf{Q}] \leq 0,\ i = 1, \ldots, 4n + 2m. \tag{5.23}$$

As mentioned in Section 5.3.3, all three methods for constraining either μ_i or ν_i to be zero are equivalent for the primal formulation. However, for the lower bound formulations in equations (5.21)–(5.23), this is not the case. The following theorem shows an ordering on these bounds.

Theorem 5.1. *Let $\underline{p}_1^\star$, $\underline{p}_2^\star$, and $\underline{p}_3^\star$ be as defined in equations* (5.21)–(5.23)*. Then*

$$\underline{p}_1^\star \geq \underline{p}_2^\star \geq \underline{p}_3^\star. \tag{5.24}$$

Proof. Let Feas($\underline{p}_1$), Feas($\underline{p}_2$), and Feas($\underline{p}_3$) denote the feasible sets for the optimization problems in equations (5.21)–(5.23) respectively. The optimizations in equations (5.21)–(5.23) have the same objective function, therefore to prove the optima ordering in equation (5.24) it is sufficient to show

$$\text{Feas}(\underline{p}_1) \subseteq \text{Feas}(\underline{p}_2) \subseteq \text{Feas}(\underline{p}_3).$$

The inequality constraints and the constraint that $Q_{11} = 1$ are the same in equations(5.22) and (5.23). The optimization in equation (5.22) has the additional single equality constraint that is not present in equation (5.23). Therefore it is clear that the right containment holds.

We now show the left containment. Let $\mathbf{Q} \in \text{Feas}(\underline{p}_1)$. Then,

$$\begin{aligned}
&\text{Tr}\left[\mathcal{P}_j \mathbf{Q}\right] = 0,\ j = 1, \ldots, n \\
\Longrightarrow &Q_{(j+n+1,j+1)} + Q_{(j+n+1,j+1)} = 0,\ j = 1, \ldots, n \\
\Longrightarrow &\sum_{j=1}^{n} Q_{(j+n+1,j+1)} + Q_{(j+n+1,j+1)} = 0 \\
\Longrightarrow &\text{Tr}\left[\mathcal{P}\mathbf{Q}\right] = 0.
\end{aligned}$$

Thus, $\mathbf{Q} \in \text{Feas}(\underline{p}_2)$, and the left containment holds. □

The inequalities in equation (5.24) can be strict for some cases. This would imply that Method 1 should be used as it provides the tightest lower bound. However, it also has the most constraints and therefore may have longer computation times. In Section 5.5, the GRI-Mech 2.11 dataset is demonstrated as a case with strict inequalities and varying computation times.

5.4 Stochastic Interpretation of the NQCQP Formulation

In Section 3.2, the rank relaxation formulation for an NQCQP is given a stochastic interpretation. This same interpretation is applied here to provide a Gaussian distribution that can be used to create a seed point for interior point (upper bound) solutions.

The optimal matrix $\mathbf{Q}^\star$ from the rank relaxation problem can be partitioned as

$$\mathbf{Q}^\star = \begin{bmatrix} 1 & \overline{\boldsymbol{\omega}}^\mathsf{T} \\ \overline{\boldsymbol{\omega}} & \boldsymbol{\Sigma} \end{bmatrix}. \tag{5.25}$$

From the stochastic interpretation, the optimal random vector is any random vector $\mathbf{\Omega}$ with first and second moments,

$$\mathrm{E}\left[\mathbf{\Omega}\right] = \overline{\boldsymbol{\omega}}, \quad \mathrm{E}\left[\mathbf{\Omega}\mathbf{\Omega}^{\mathsf{T}}\right] = \mathbf{\Sigma}. \tag{5.26}$$

We will use a Gaussian distribution with these statistics, $\mathbf{\Omega} \sim \mathcal{N}(\overline{\boldsymbol{\omega}}, \mathbf{\Sigma} - \overline{\boldsymbol{\omega}}\overline{\boldsymbol{\omega}}^{\mathsf{T}})$.

To solve for an upper bound on the NQCQP formulation, a sample from this distribution will tend to provide a good seed point for an interior point solver (e.g. MATLAB's `fmincon`). However, since the NQCQP formulation of the ℓ^1-norm approximation has twice as many variables as the original ℓ^1-norm approximation problem, it is computationally more efficient to solve the original problem. To seed this problem, we simply transform the sampled point back into the $\mathbf{x}$-space. First partition the sampled point $\boldsymbol{\omega}_{\text{seed}}$,

$$\boldsymbol{\omega}_{\text{seed}} = \begin{bmatrix} \boldsymbol{\mu}_{\text{seed}} \\ \boldsymbol{\nu}_{\text{seed}} \end{bmatrix}, \tag{5.27}$$

then transform back to the $\mathbf{x}$-space,

$$\mathbf{x}_{\text{seed}} = \boldsymbol{\mu}_{\text{seed}} - \boldsymbol{\nu}_{\text{seed}} + \mathbf{x}_0. \tag{5.28}$$

5.5 Example: GRI-Mech Dataset

For the GRI-Mech release 3.0, researchers wanted to allow only a small number of the 102 parameters to vary from their release 2.11 "nominal" values. After much deliberation and very careful planning they chose 31 parameters to free from the nominal values for optimization [106, 120]. These were mostly chosen with domain knowledge of the parameters and the experiments used to determine their nominal value [41]. Because of the many possible combinations of the 102 parameters, the choice of parameters took months to come to. The finished optimized result, the GRI-Mech 3.0 nominal, is not in the feasible set, and therefore isn't consistent with all of the experimental data, uncertainty, and models.

Using the ℓ^1-norm approximation to the minimal cardinality problem derived in this chapter, we found a feasible point that only requires 52 parameters to deviate from the GRI-Mech 2.11 nominal value (to within a tolerance of 0.01%). Of the 31 parameters optimized for GRI-Mech 3.0, 29 are in the list of parameters that deviated. This means, in significantly less time, we are able to suggest a list of 53 parameters to consider for optimization that contain most of the parameters eventually chose by the GRI-Mech researchers themselves.

To obtain this result, the lower bounds on the optimal one-norm difference were first computed. Table 5.1 shows the bounds computed using the different equality constraint formulations discussed in Section 5.3.3 as well as computation times.

Table 5.1: Objective lower bounds and computation times for GRI-Mech 2.11 minimum ℓ^1-norm distance. The three lower bounds correspond to the three different equality constraint formulations described in Section 5.3.3.

Formulation	Bound	CPU Time (sec)
S-proc. w/ method 1	30.18	17.15
S-proc. w/ method 2	29.64	42.82
S-proc. w/ method 3	29.00	37.58

The three lower bounds exhibit the ordering proven in Theorem 5.1 and provide an example where the inequalities in equation (5.24) are strict. Despite having the most constraints, the bound using equality constraint method 1 took the significantly less time than the other methods.

Upper bounds for the solution were then computed using several different seeds for `fmincon`. The interior point solution also provides the desired feasible point. Table 5.2 provides the bounds and computation times. The inner bound was computed using several different seedings. It appears that solutions do not really depend on the seeding; however, there was a slight difference in computation time, although maybe not enough to be of any consequence.

Table 5.2: Objective upper bounds and computation times for GRI-Mech 2.11 minimum ℓ^1-norm distance. The seed point for each problem was either $\mathbf{0}$, or sampled from the distribution provided by the rank relaxation solution using one of the three equality constraint methods.

Seed	$\|\mathbf{x}-\mathbf{x}_0\|_1$	$\mathrm{card}(\mathbf{x}-\mathbf{x}_0)$	CPU Time (sec)
$\mathbf{0}$	33.03	52	34.18
via method 1	33.03	52	30.89
via method 2	33.03	52	35.34
via method 3	33.03	52	33.16

The best lower bound implies that the upper bound solution (which provides a feasible parameter vector) cannot get much better in terms of the ℓ^1-norm objective. Trying to find a better solution by actually solving the cardinality problem, is not realistic. The number of combinations of exactly 51 parameters (one less deviation than we found) is about 4×10^{29}. Checking if there is a solution with 51 deviated parameters would, in the worst case, require solving a feasibility problem for each combination.

Chapter 6

Active Subspace Discovery

Many applications and analysis techniques in science and engineering use simulation as a means for gauging a system's behavior. In coarse designs, simulation provides a qualitative look at system attributes. Much denser simulation might be required for more quantitative assessments. However, in advanced simulation techniques with complex models, each simulation can take hours or days. Processes with many variables require many simulation runs to adequately cover the space of responses. In this case, even if each simulation only takes a few minutes, the total simulation time can grow exponentially.

Surrogate modeling is the technique of creating an algebraic approximation to the simulation's map from parameters to response. The resulting *response surface* or *surrogate model* is much more efficient to evaluate than the original simulation and can provide much insight into the behavior of the original system [75]. The best form for a surrogate model depends on the application. They are formed with various methods such as standard regression, support vector machines [107], and kriging methods [79]. Data Collaboration techniques use quadratic surrogate models to calculate outer bounds to various optimization objectives such as the consistency measure, response prediction, and representative feasible parameter (see §2.4 and Chapters 3 and 5). Ryan Feeley derived methods for using rational quadratic surrogates with the Data Collaboration techniques, as well as the technique of using intermediate surrogates [34].

One of the major problems facing surrogate modeling occurs when long simulation time is required to adequately sample the model response. To fit a surrogate model, many simulation runs are often required. The number of simulations depends on the form of the surrogate (i.e. the number of basis functions) and the dimension of the parameter vector. For example, with a quadratic surrogate, the number of basis functions (monomials with a degree of at most 2) increases quadratically with the number of parameters.

This chapter describes a technique for discovering the possible dependence of the response to an lower-dimensional active subspace of the parameters. If such an active subspace were known, the amount of simulation required to make a surrogate would depend on the subspace dimension instead of the original "full" dimension.

Subspace dependence is not a new concept. However, the focus of many works is on searching for a subspace dependence of the multivariate output of a function or of the evolving state vector of a set of coupled ODEs [8, 51, 59]. Some works use local subspace dependence to preserve neighborhood relationships and fit low-dimensional nonlinear manifolds to the data [25, 45, 94]. High-dimensional model representations (HDMRs) build up a surrogate model by iteratively fitting along every coordinate-aligned subspace starting with the 0-dimensional subspace.[1] and stopping after reasonable errors have been achieved [87, 88]

The following sections will derive the technique for active subspace discovery and analyze the computational requirements for discovering it.

6.1 Active Subspace Dependence

Let f be a scalar-valued function of n active variables. Our goal is to find a low-rank active subspace of the input variables that is the main contributor to the variation of f. In other words, we would like to discover an $n \times r$ matrix $\mathbf{S}$, with $r < n$, and a function $g : \mathbb{R}^r \to \mathbb{R}$ such that f is well approximated as

$$f(\mathbf{x}) \approx g(\mathbf{S}^\mathsf{T}\mathbf{x}). \tag{6.1}$$

The quality of the approximation requires that f varies mostly along an *active subspace* of the original variables. In the current notation the active subspace is the range space of the columns of $\mathbf{S}$, written $\mathcal{R}(\mathbf{S})$.

Approximations of the type shown in equation (6.1) can always be made. A trivial example is seen by setting $\mathbf{S}$ to the identity matrix (r equals n) and setting the function g to be equivalent to f. However, the goal is to find a good approximation such that r is as small as possible. This chapter focuses on finding a suitable matrix $\mathbf{S}$. Chapter 7 will focus on designing experiments for fitting the function g.

The function g and matrix $\mathbf{S}$ in equation (6.1) are not unique for an approximation. Take for example any $r \times r$ invertible matrix $\mathbf{R}$, and define $tildeg := g \circ \mathbf{R}^{-1}$ and $\tilde{\mathbf{S}} := \mathbf{S}\mathbf{R}^\mathsf{T}$. Replacing g with $\tilde{g}$ and $\mathbf{S}$ with $\tilde{\mathbf{S}}$ does not change the approximation.

$$\begin{aligned} g(\mathbf{S}^\mathsf{T}\mathbf{x}) &= g(\mathbf{R}^{-1}\mathbf{R}\mathbf{S}^\mathsf{T}\mathbf{x}) \\ &= \tilde{g}(\tilde{\mathbf{S}}^\mathsf{T}\mathbf{x}). \end{aligned} \tag{6.2}$$

It is for this reason that when searching for an appropriate $\mathbf{S}$ matrix, the objective is not to find a specific $\mathbf{S}$, but rather one that spans the appropriate subspace of $\mathbb{R}^n$,

[1] This is specifically referred to as the cut-HDMR.

namely, that on which the function depends most. It is simple to see that since $\mathbf{R}$ is invertible, $\mathcal{R}(\mathbf{S})$ and $\mathcal{R}(\tilde{\mathbf{S}})$ are equal.

Claim 6.1. $\mathcal{R}(\mathbf{S}) = \mathcal{R}(\tilde{\mathbf{S}})$.

Proof. First see that $\mathcal{R}(\mathbf{S}) \subseteq \mathcal{R}(\tilde{\mathbf{S}})$. Let $\mathbf{x} \in \mathcal{R}(\mathbf{S})$. Then there exists a $\mathbf{y} \in \mathbb{R}^r$ such that $\mathbf{x} = \mathbf{S}\mathbf{y}$. Since $\mathbf{R}$ is invertible, there exists a $\mathbf{z} \in \mathbb{R}^n$ such that $\mathbf{y} = \mathbf{R}^\mathsf{T}\mathbf{z}$ and thus $\mathbf{x} = \mathbf{S}\mathbf{R}^\mathsf{T}\mathbf{z} = \tilde{\mathbf{S}}\mathbf{z}$. Therefore $\mathbf{x} \in \mathcal{R}(\tilde{\mathbf{S}})$.

Now see that $\mathcal{R}(\mathbf{S}) \supseteq \mathcal{R}(\tilde{\mathbf{S}})$. Let $\mathbf{z} \in \mathcal{R}(\tilde{\mathbf{S}})$. Then there exists a $\mathbf{y} \in \mathbb{R}^r$ such that $\mathbf{z} = \tilde{\mathbf{S}}\mathbf{y} = \mathbf{S}\mathbf{R}^\mathsf{T}\mathbf{y}$. Define $\mathbf{x} = \mathbf{R}^\mathsf{T}\mathbf{y}$. Then $\mathbf{z} = \mathbf{S}\mathbf{x}$. Therefore $\mathbf{z} \in \mathcal{R}(\mathbf{S})$. □

For simplicity we assume that the columns of $\mathbf{S}$ are orthonormal and therefore $\mathbf{S}^\mathsf{T}\mathbf{S} = \mathbf{I}_r$. The non-uniqueness of $\mathbf{S}$ allows this choice.

6.2 Methodology

The key to discovering the active subspace of the function f is noting that if the factorization in equation (6.1) holds and the function is differentiable at a point $\mathbf{x}$, then the gradient of f at $\mathbf{x}$ multiplicatively factors via the chain rule of derivatives as

$$\nabla f(\mathbf{x}) = \nabla g(\mathbf{S}^\mathsf{T}\mathbf{x}) \cdot \mathbf{S}^\mathsf{T}, \tag{6.3}$$

where $\nabla f(\mathbf{x})$ is the $1 \times n$ gradient vector of the function f at the point $\mathbf{x}$. Furthermore, this factorization holds when the gradients at many points are stacked into a matrix. Define the matrices

$$\mathbf{F} := \begin{bmatrix} \nabla f(\mathbf{x}_1) \\ \vdots \\ \nabla f(\mathbf{x}_N) \end{bmatrix}, \quad \mathbf{G} := \begin{bmatrix} \nabla g(\mathbf{S}^\mathsf{T}\mathbf{x}_1) \\ \vdots \\ \nabla g(\mathbf{S}^\mathsf{T}\mathbf{x}_N) \end{bmatrix} \tag{6.4}$$

where each $\mathbf{x}_k$ is an n-dimensional vector at a different location in the domain. Then

$$\mathbf{F} = \mathbf{G}\mathbf{S}^\mathsf{T}. \tag{6.5}$$

Note that in general, the rank of the matrix $\mathbf{F}$ is the number of gradient locations, N, if $N < r$ and is r if $r \geq N$. For this factorization to be defined, the function f needs to be differentiable at each $\mathbf{x}_k$; however, it need not be differentiable everywhere.

The rank of $\mathbf{F}$ naturally lends itself to an iterative algorithm. As more gradients are computed and added to $\mathbf{F}$, the rank should continue to grow. As soon as the rank stops growing, a matrix factorization will provide the appropriate linear transformation matrix $\mathbf{S}$. This matrix factorization is easily computed with a singular value decomposition (SVD). Examine the singular value decomposition of the matrix $\mathbf{F}$

$$\mathbf{F} = \begin{bmatrix} \mathbf{U}_1 \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^\mathsf{T} \\ \mathbf{V}_2^\mathsf{T} \end{bmatrix}. \tag{6.6}$$

The factorization is then written by setting $\mathbf{G} = \mathbf{U}_1\mathbf{\Sigma}_{11}$ and $\mathbf{S} = \mathbf{V}_1$.

In practice, $\mathbf{F}$ has full rank; however, the singular values of $\mathbf{F}$ may decay rapidly. Given this, the active subspace is defined as the span of the directions associated with the largest singular values of $\mathbf{F}$. The threshold for deciding which singular values are large enough is a parameter in the active subspace discovery algorithm.

The procedure for discovering the active subspace is shown in Algorithm 6.1.

Algorithm 6.1 Active Subspace Discovery

Require: f {function of interest}
Require: $\mathcal{H}$ {n-dimensional hyperrectangle domain for f}
Require: τ {Singular value cutoff threshold, 0.1 or 0.05 recommended}
1: $n \leftarrow$ dimension of $\mathcal{H}$.
2: Initialize matrix $\mathbf{F}$ (empty)
3: $\{\texttt{dloc}_k\}_{k=1}^{n} \leftarrow$ n-point, n-dimensional Latin hypercube design on $\mathcal{H}$.
4: **for** $k = 1$ to n **do**
5: $\quad\texttt{grad} \leftarrow$ gradient row vector of f at location $\texttt{dloc}_k$
6: $\quad$Add `grad` to bottom of matrix $\mathbf{F}$.
7: $\quad\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} \leftarrow$ SVD of $\mathbf{F}$ {such that $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathsf{T}}$}
8: $\quad\boldsymbol{\sigma} \leftarrow \text{diag}(\mathbf{\Sigma})$
9: $\quad$**if** $k > 1$ **and** there exists an i such that $\sigma_{i+1} \leq \tau\sigma_i$ **then**
10: $\quad\quad r_j \leftarrow$ the minimum i such that $\sigma_{i+1} \leq \tau\sigma_i$
11: $\quad\quad\mathbf{S} \leftarrow$ first r columns of $\mathbf{V}$
12: $\quad\quad$**return** $\mathbf{S}$ {Algorithm exits.}
13: $\quad$**end if**
14: **end for**
15: $\mathbf{S} \leftarrow \mathbf{V}$ {Only executes if no active subspace found yet.}
16: **return** $\mathbf{S}$

The locations where the gradients are computed should be spread out in the domain of the function. This algorithm uses a Latin hypercube sample design to choose these locations (line 3 of Algorithm 6.1). Latin hypercube sample designs are set up to be spread out in each variable. This is done by setting up a classic Latin square in n-dimensions (examining only one treatment) [70] and uniformly choosing a point in each of the chosen sub-cubes. Each point in a Latin hypercube sample design is uniformly distributed on the hypercube, but the points are not independent. This sample ensures that each variable is represented in a fully stratified manner [67].

The method of computation of gradients (line 5 of Algorithm 6.1) varies from application to application. In some applications the local sensitivities are available with a single simulation run. Often, $n + 1$ simulations near the gradient location are computed to calculate a numerical derivative. If the simulations have noise in their output, it may be necessary to run more simulations to average out error in the derivative.

The algorithm is an iterative one, that completes as soon as the singular value

threshold has been crossed (line 9 of Algorithm 6.1). If an appropriate subspace matrix $\mathbf{S}$ is discovered, the for-loop does not complete. This is done to reduce the total number of simulation runs. However, the accuracy of the active subspace generally increases with the number of gradients computed (see §6.7). Therefore, if gradients are cheap to compute for particular problems (e.g. requiring only one simulation run) it makes sense to compute the gradient at all n locations.

A potential problem arises with the algorithm when f is nonlinear and has a region of the domain with nonzero measure where the gradient is constant. When this situation occurs, there is a nonzero probability that a new gradient calculation will be the same as a previous one, and therefore the rank of $\mathbf{F}$ will not increase. This would yield a subspace dimension that is potentially smaller than the actual active subspace. The same situation would occur if there is a nonzero measure region where there is a constant gradient that is equal to the multiplicative scaling of a gradient already calculated. For example, take $f(x) = \|\mathbf{S}^\mathsf{T}\mathbf{x}\|_1$ for some fixed $n \times r$ matrix $\mathbf{S}$. On each orthant, the gradient of f is constant. Moreover, $\nabla f(\mathbf{x}) = -\nabla f(-\mathbf{x})$ and hence the gradients at $\mathbf{x}$ and at $-\mathbf{x}$ are linearly dependent.

6.3 Complexity Analysis

The limiting factor in the time-complexity of the active subspace discovery algorithm is the number of evaluations of the function f and likewise the computation of gradients. Let κ be the number of simulations required to compute a single gradient of f or the equivalent amount of time. For example, if gradients are calculated numerically, with no noise in f, then $n+1$ tightly bunched simulations are needed, and hence $\kappa = n + 1$. If a gradient requires only a single simulation, but one that takes twice as long as usual, $\kappa = 2$. If the underlying rank is r, the algorithm must run $r+1$ iterations to notice a drop-off in the singular values. Therefore the time T_{subspace} that it takes to discover an r dimensional active subspace is

$$T_{\text{subspace}} \in O\left(\kappa \cdot (r+1)\right), \tag{6.7}$$

where *big-O* notation[2] is used to upper bound the time [105].

Presumably, f is evaluated at least once at every location the gradient is computed. This simulation can be reused to help fit a surrogate model to g. If multiple evaluations of f are required to compute a gradient at each location, only one of the simulations is useful for surrogate fitting because the points will be bunched together. Therefore the $r+1$ simulation results can be reused for fitting.

Let $\varphi(r)$ be the number of evaluations of f required to make a surrogate fit in r dimensions. This will depend on the form of the surrogate function (linear, quadratic,

[2]Let g be a function $g : \mathbb{N} \to \mathbb{R}_+$. $O(g(n))$ is the set of all functions $f : \mathbb{N} \to \mathbb{R}_+$ such that there exists positive integers c and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. In other words, $O(g(n))$ is the set of all functions that are asymptotically upper bounded by g [105].

rational quadratic, etc). $\varphi(r)$ also depends on the behavior of the actual function f. If an evaluation of f is noisy or if it has small-amplitude high-order effects, more points are required to capture the behavior of the function. The time $T_{\text{fit}-\text{full}}$ to fit a surrogate model to f in the full n dimensions is

$$T_{\text{fit}-\text{full}} \in O\left(\varphi(n)\right). \tag{6.8}$$

The total time $T_{\text{fit}-\text{subspace}}$ to fit a surrogate model in the subspace dimension including discovery of the subspace is

$$T_{\text{fit}-\text{subspace}} \in O\left(\varphi(r) + (\kappa - 1)(r + 1)\right). \tag{6.9}$$

The subspace discovery is beneficial time-wise if $T_{\text{fit}-\text{full}} > T_{\text{fit}-\text{subspace}}$. It is likely that fitting error will increase due to the use of a subspace, as governed by the threshold τ on line 9 in Algorithm 6.1. If subspace discovery is to be useful, a significant time decrease is required. Chapter 7 will further explore the design of evaluation points in the active subspace and subsequent surrogate fitting.

As an example, say that the surrogate form of interest is a quadratic and that $\varphi(n) = (n + 1)(n + 2)/2$, which is the number of coefficients in a n-dimensional quadratic. Furthermore, say numerical gradients are used such that $\kappa = n + 1$. If $n = 60$ and the underlying active subspace has dimension $r = 5$, then 1,891 evaluations are needed to fit the quadratic in all 30 variables. Only 381 evaluations in total are needed to discover the subspace and fit the quadratic in 5 variables.

6.4 Relationship to Principal Component Analysis

Principal component analysis (PCA) is the process by which linear transformations of correlated variables are generated that produce relatively uncorrelated variables [37]. Given a set of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$, PCA provides a rotation to a set of orthogonal ordered coordinates such that the first coordinate has the most variability, the second coordinate has the second most variability and so on. These new directions are called the *principal components*. PCA is often used for reduction of a set of data's dimensionality by using only the top few principal components.

In general, PCA cannot by used directly to find the active subspace of a function. The input vector $\mathbf{x}$ is assumed to have uncorrelated components. On a set of these input vectors drawn from a uniform sample on a hypercube domain $\mathcal{H}$, PCA would find that the variables are uncorrelated (on average) and no direction has any more variability than the others. In general, f is nonlinear and can have arbitrary variability.

The subspace discovery algorithm presented in Section 6.2, however, is a principal component analysis in the gradient space. If the function f only depends on an r dimensional subspace, then there will be directions of high variability in the gradient space and, maybe more importantly, directions of low variability. The columns of the subspace matrix $\mathbf{S}$ are the first r principal components of the gradient vectors.

6.5 Measure of Subspace Containment

This section describes a simple measure of one subspace's containment of another, and how that limits the ability to fit a quadratic function. The measure is found in [81]. Suppose $\mathcal{L}$ and $\mathcal{D}$ are two subspaces of $\mathbb{R}^n$. Define a *measure of containment of* $\mathcal{L}$ *in* $\mathcal{D}$ as

$$m(\mathcal{L},\mathcal{D}) := \max_{\mathbf{l}\in\mathcal{L},\|\mathbf{l}\|_2\leq 1} \|\mathbf{l}-\Pi_{\mathcal{D}}(\mathbf{l})\|_2 \tag{6.10}$$

where $\Pi_{\mathcal{D}}$ is the orthogonal projection onto $\mathcal{D}$. Note that if $\mathcal{L}\subseteq\mathcal{D}$, then $m(\mathcal{L},\mathcal{D})=0$, and if $\mathcal{L}$ contains vectors orthogonal to $\mathcal{D}$, then $m(\mathcal{L},\mathcal{D})=1$.

Let $\mathbf{L}$ be a matrix whose orthonormal columns (so $\mathbf{L}^\mathsf{T}\mathbf{L}=\mathbf{I}$) span $\mathcal{L}$ and similar for $\mathbf{D}$. Also let $\mathbf{D}_\perp$ be a matrix whose orthonormal columns span the orthogonal complement of $\mathcal{D}$. Note that the projection operator $\Pi_{\mathcal{D}}$ is just matrix multiplication by $\mathbf{D}\mathbf{D}^T$. Therefore

$$\begin{aligned} m(\mathcal{L},\mathcal{D}) &= \max_{\boldsymbol{\eta},\|\mathbf{L}\boldsymbol{\eta}\|_2\leq 1} \left\|\mathbf{L}\boldsymbol{\eta}-\mathbf{D}\mathbf{D}^\mathsf{T}\mathbf{L}\boldsymbol{\eta}\right\|_2 \\ &= \max_{\boldsymbol{\eta},\|\boldsymbol{\eta}\|_2\leq 1} \left\|\left(I-\mathbf{D}\mathbf{D}^\mathsf{T}\right)\mathbf{L}\boldsymbol{\eta}\right\|_2 \\ &= \max_{\boldsymbol{\eta},\|\boldsymbol{\eta}\|_2\leq 1} \left\|\mathbf{D}_\perp\mathbf{D}_\perp^\mathsf{T}\mathbf{L}\boldsymbol{\eta}\right\|_2 \\ &= \bar{\sigma}\left[\mathbf{D}_\perp\mathbf{D}_\perp^\mathsf{T}\mathbf{L}\right] \\ &= \bar{\sigma}\left[\mathbf{D}_\perp^\mathsf{T}\mathbf{L}\right]. \end{aligned} \tag{6.11}$$

Next, consider a quadratic form $\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{S}\mathbf{L}^\mathsf{T}\mathbf{x}$ which is to be approximated by a quadratic form $\mathbf{x}^\mathsf{T}\mathbf{D}\mathbf{B}\mathbf{D}^\mathsf{T}\mathbf{x}$, with $\mathbf{D}=\mathbf{D}^\mathsf{T}$ given, and $\mathbf{B}=\mathbf{B}^\mathsf{T}$ to be determined.

For a given $\mathbf{B}$, the maximum singular value gives the maximum error over the unit ball,

$$\max_{\|x\|_2\leq 1} \left|\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{x}-\mathbf{x}^\mathsf{T}\mathbf{D}\mathbf{B}\mathbf{D}^\mathsf{T}\mathbf{x}\right| = \bar{\sigma}\left[\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}-\mathbf{D}\mathbf{B}\mathbf{D}^\mathsf{T}\right]. \tag{6.12}$$

The goal of the approximation is to minimize this error given a choice of $\mathbf{B}$. This is easily solved with a series of manipulations.

$$\begin{aligned} &\min_{\mathbf{B}}\ \bar{\sigma}\left(\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}-\mathbf{D}\mathbf{B}\mathbf{D}^\mathsf{T}\right) \\ &= \min_{\mathbf{B}}\ \bar{\sigma}\left(\begin{bmatrix}\mathbf{D}^\mathsf{T}\\ \mathbf{D}_\perp^\mathsf{T}\end{bmatrix}\left(\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}-\mathbf{D}\mathbf{B}\mathbf{D}^\mathsf{T}\right)\begin{bmatrix}\mathbf{D} & \mathbf{D}_\perp\end{bmatrix}\right) \\ &= \min_{\mathbf{B}}\ \bar{\sigma}\left(\begin{bmatrix}\mathbf{D}^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}-\mathbf{B} & \mathbf{D}^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}_\perp\\ \mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D} & \mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}_\perp\end{bmatrix}\right) \\ &= \min_{\mathbf{X}}\ \bar{\sigma}\left(\begin{bmatrix}\mathbf{X} & \mathbf{D}^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}_\perp\\ \mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D} & \mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}_\perp\end{bmatrix}\right) \\ &= \max\left\{\bar{\sigma}\left(\begin{bmatrix}\mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D} & \mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}_\perp\end{bmatrix}\right),\ \bar{\sigma}\left(\begin{bmatrix}\mathbf{D}^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}_\perp\\ \mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}_\perp\end{bmatrix}\right)\right\} \\ &= \max\left\{\bar{\sigma}\left(\mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\right),\ \bar{\sigma}\left(\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{D}_\perp\right)\right\} \\ &= \bar{\sigma}\left(\mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\right). \end{aligned} \tag{6.13}$$

Note that $\bar{\sigma}\left[\mathbf{D}_\perp^\mathsf{T}\mathbf{L}\mathbf{A}\right] \leq \bar{\sigma}\left[\mathbf{D}_\perp^\mathsf{T}\mathbf{L}\right]\bar{\sigma}\left[\mathbf{A}\right]$. Moreover $\bar{\sigma}\left[\mathbf{A}\right] = \max_{\|\mathbf{x}\|_2\leq 1}\left|\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{x}\right|$. Therefore,

$$\min_{\mathbf{B}}\max_{\|\mathbf{x}\|\leq 1}\left|\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{x} - \mathbf{x}^\mathsf{T}\mathbf{D}\mathbf{B}\mathbf{D}^\mathsf{T}\mathbf{x}\right| \leq m(\mathcal{L},\mathcal{D})\max_{\|\mathbf{x}\|\leq 1}\left|\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{A}\mathbf{L}^\mathsf{T}\mathbf{x}\right|. \tag{6.14}$$

So (as expected) the extent to which the actual subspace ($\mathcal{L}$) is contained in the approximating subspace ($\mathcal{D}$) limits the achievable approximation error. For the toy examples in the following section, the active subspace is known *a priori*, and this measure will give a sense of how well the subspace discovery algorithm is working.

6.6 Examples

This section will demonstrate the active subspace discovery algorithm on several example systems. The first two will be contrived toy examples, where the actual active subspaces are known for comparison. The other examples will be from real systems where the active subspaces are unknown *a priori*.

6.6.1 Toy Functions

We chose an arbitrary 30×5 matrix $\mathbf{S}$ such that $\mathbf{S}^\mathsf{T}\mathbf{S} = \mathbf{I}_3$. If $\mathbf{z} = \mathbf{S}^\mathsf{T}\mathbf{x}$, then let $f_1(\mathbf{x}) = g_1(\mathbf{S}^\mathsf{T}\mathbf{x})$ where g_1 is defined as

$$g_1(\mathbf{z}) = \sin(z_1) + z_2 z_3 z_5 + z_1 z_4^2 + e^{z_3}\cos(z_5^3). \tag{6.15}$$

The domain of f_1 is $\mathbf{x} \in [-1,\ 1]^{30}$.

First we examine the sensitivity coefficients for the 30-dimensional space. Here, sensitivity analysis refers to finding a relative sense of the function's dependence on its parameters [58]. These were computed by evaluating the function on a fractional factorial design from a Hadamard matrix [48]. An affine function is fitted to these evaluations. The magnitude of the linear coefficients of the fit act as the sensitivity coefficients. Figure 6.1 shows these sensitivity coefficients sorted by magnitude. From the figure, we see most of the 30 variables are active.

Assuming that we do not have direct access to the gradient of f_1, the gradients are calculated numerically. After 6 iterations of the subspace discovery algorithm, the singular values of the gradient matrix are as shown in Figure 6.2. This results in a 5 dimensional subspace, which required 186 function evaluations to compute. The measure of containment of the "real" subspace in the subspace approximation is 0.032, and therefore the approximate active subspace is very close to the true one.

As another example, take a function f_2 that depends mostly, but not perfectly, on a 5 dimensional subspace. Again an arbitrary 30×5 matrix $\mathbf{S}$ with $\mathbf{S}^\mathsf{T}\mathbf{S} = \mathbf{I}_3$ is

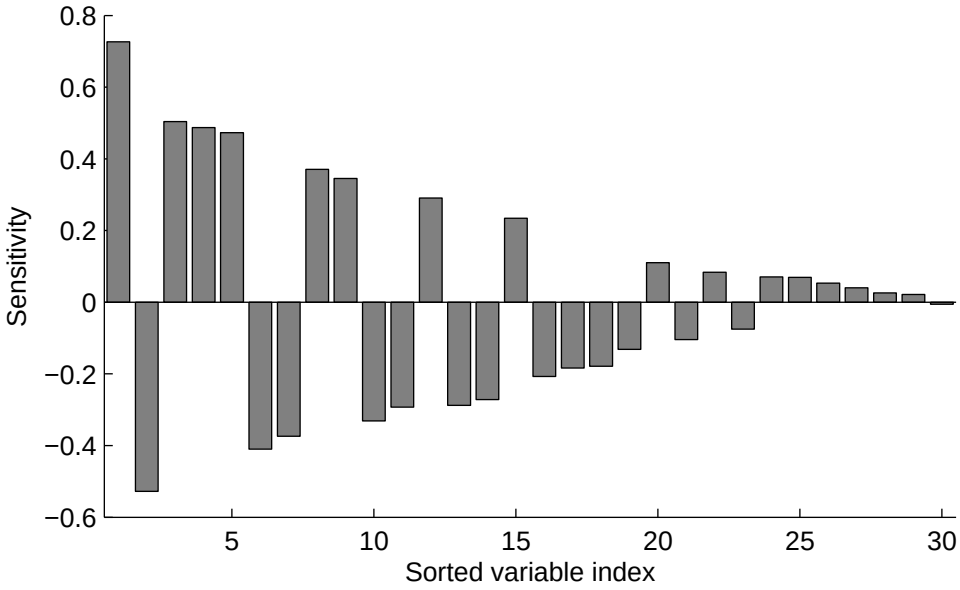

Figure 6.1: Sensitivity coefficients of f_1 from a Hadamard design. Most variables are active.

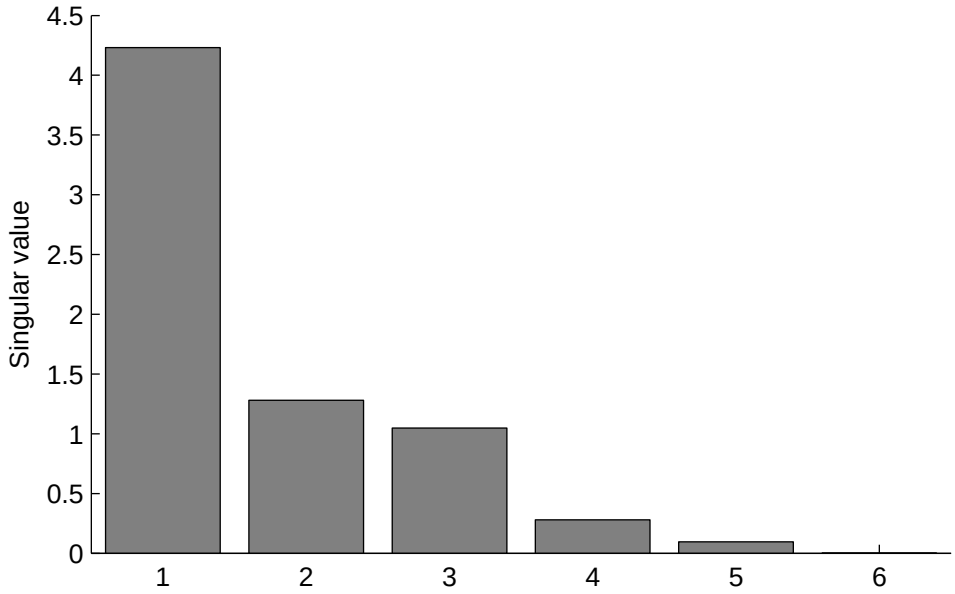

Figure 6.2: Singular values of the gradient matrix for f_1, after 6 iterations of the subspace discovery algorithm. Active subspace appears to be either 5- or 6-dimensional depending on the threshold.

chosen and fixed. Let $\mathbf{S}_\perp$ be the 30×25 matrix with orthonormal columns such that $\mathcal{R}(\mathbf{S}) \perp \mathcal{R}(\mathbf{S}_\perp)$, i.e. $[\mathbf{S}\,\mathbf{S}_\perp]$ is unitary. Let f_2 be a quadratic function of the form

$$f_2(\mathbf{x}) = \left[\begin{smallmatrix}1\\ \mathbf{S}^\mathsf{T}\mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Q}_1 \left[\begin{smallmatrix}1\\ \mathbf{S}^\mathsf{T}\mathbf{x}\end{smallmatrix}\right] + \frac{1}{500}\left[\begin{smallmatrix}1\\ \mathbf{S}_\perp^\mathsf{T}\mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Q}_2 \left[\begin{smallmatrix}1\\ \mathbf{S}_\perp^\mathsf{T}\mathbf{x}\end{smallmatrix}\right]. \tag{6.16}$$

In this case, f_2 does not vary solely on a 5-dimensional subspace, but does mostly. This is why $\mathcal{R}(\mathbf{S})$ is called the *active* subspace.

Again, most or all of the 30 variables are considered active (see Figure 6.3), and the singular values from the subspace discovery algorithm imply there is a 5-dimensional active subspace (see Figure 6.4). This took 186 function evaluations to compute. The measure of containment of the "real" active subspace in the subspace approximation is 0.041, and hence the approximation is good.

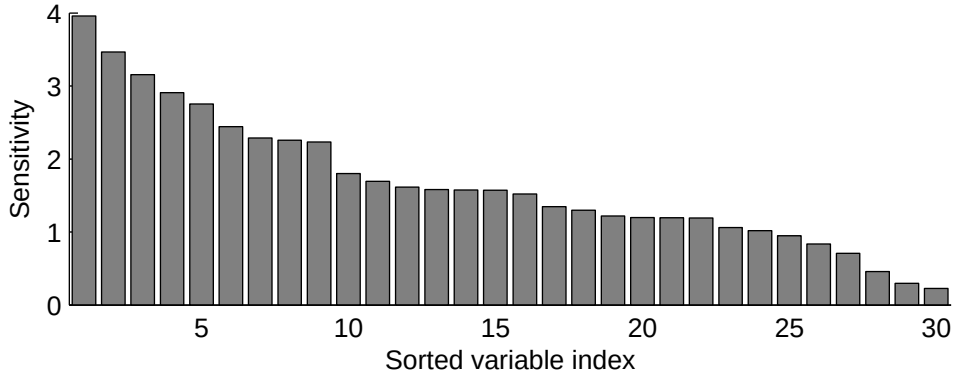

Figure 6.3: Sensitivity coefficients of f_2 from a Hadamard design. Most variables are active.

6.6.2 Logic Circuit

We are given an electric circuit design to simulate its behavior. For each circuit element (resistor, capacitor, transistor, etc.) there exists models relating currents and voltages given some parameters. Due to manufacturing uncertainties the parameters are unknown but have a nominal value and some uncertainty bounds. We are interested in the behavior of a scalar attribute of the circuit's output in response to changes in these parameters.

The simulation tool SPICE [112] can provide sensitivity information (local derivatives with respect to the parameters) by creating an "adjoint" circuit where the voltages are the sensitivities. The adjoint system is more complex and takes about twice as long to simulate as a regular simulation [110]. However, as of now this feature is only available in some commercial releases of SPICE to which we do not have access.

Figure 6.5 shows the circuit diagram of a simple adder element used in multipliers. The 10 gates shown are all simple logic gates each consisting of two transistors (a

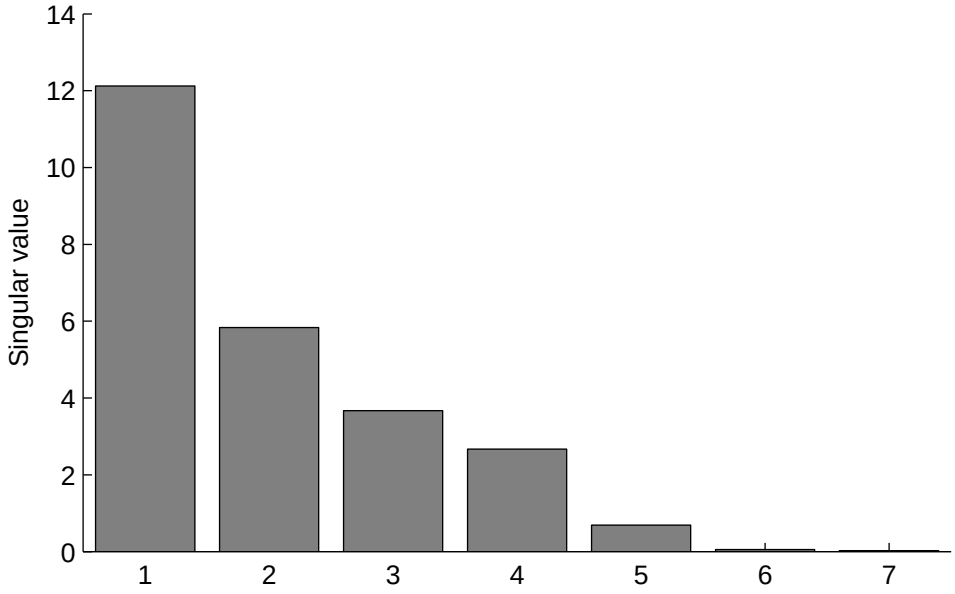

Figure 6.4: Singular values of the gradient matrix for f_2, after 6 iterations of the subspace discovery algorithm. Active subspace appears to be either 5- or 6-dimensional depending on the threshold.

p-type and an n-type). Each transistor has 3 uncertain parameters associated with it. Therefore, there are a total of 60 parameters. For this example, the simulation is setup with I1, I2, I4, and Cin set to high (Vdd), and I3 set to a rising step. The function of interest will be the map from the 60 parameters to the voltage of node S (the sum signal) at time $t = 5.43$ns (about halfway through the step transition). The domain of the parameters is bounded by plus and minus 15% of their nominal values.

The variable sensitivities for the adder signal function are shown in Figure 6.6. The singular values from the first 20 iterations of the subspace discovery algorithm (Figure 6.7) show that the active subspace may be as small as one-dimensional. Had a one-dimensional active subspace been chosen after only 2 iterations, only 122 simulations would have been required. If we had access to a simulator that provided sensitivities with only one simulation, only two iterations would have been needed to find the active subspace.

6.6.3 Methane Combustion

GRI-Mech 3.0 [106] target CH3.C1a is the maximum CH3 concentration in a shock tube oxidation of methane [18]. The GRI-Mech project included quadratic surrogate fits, but these were originally generated from shock-tube simulation codes. The simulation has 313 parameters. The sensitivities of these parameters is shown in Figure 6.8. The surrogate fit for the GRI-Mech project uses the top 11 ranked

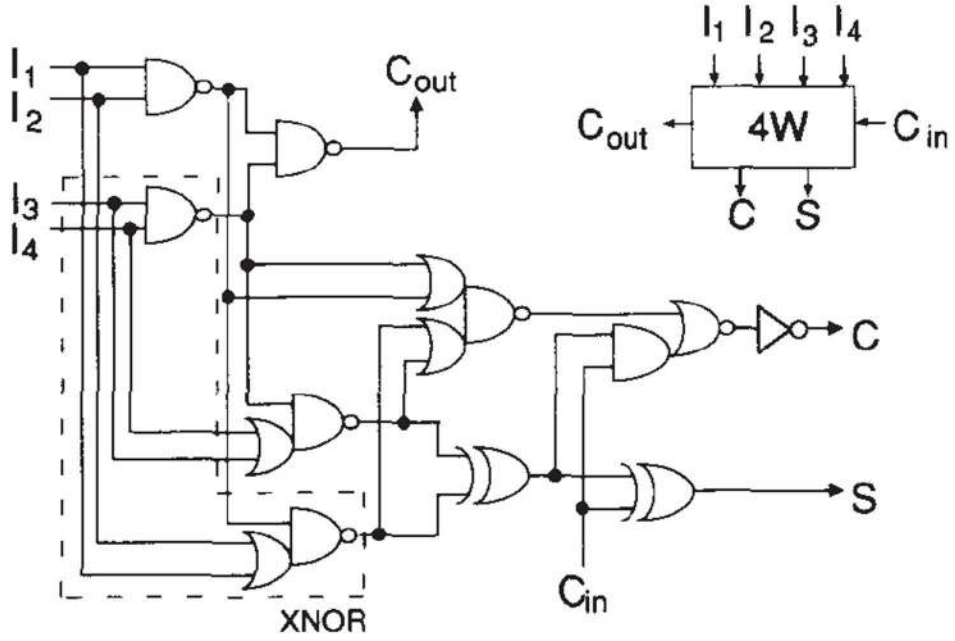

Figure 6.5: Adder element used in multipliers. Graphic from IEEE Journal of Solid-State Circuits [49].

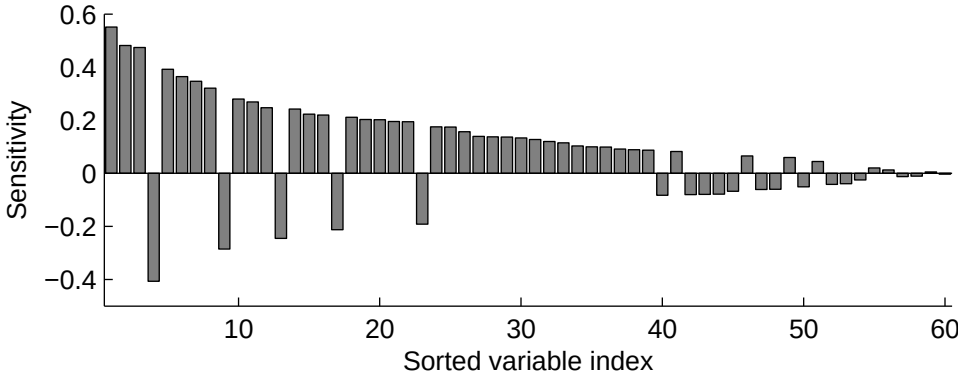

Figure 6.6: Sensitivity coefficients of node S at $t = 5.43$ns in the adder element circuit from a Hadamard design. Most variables are active.

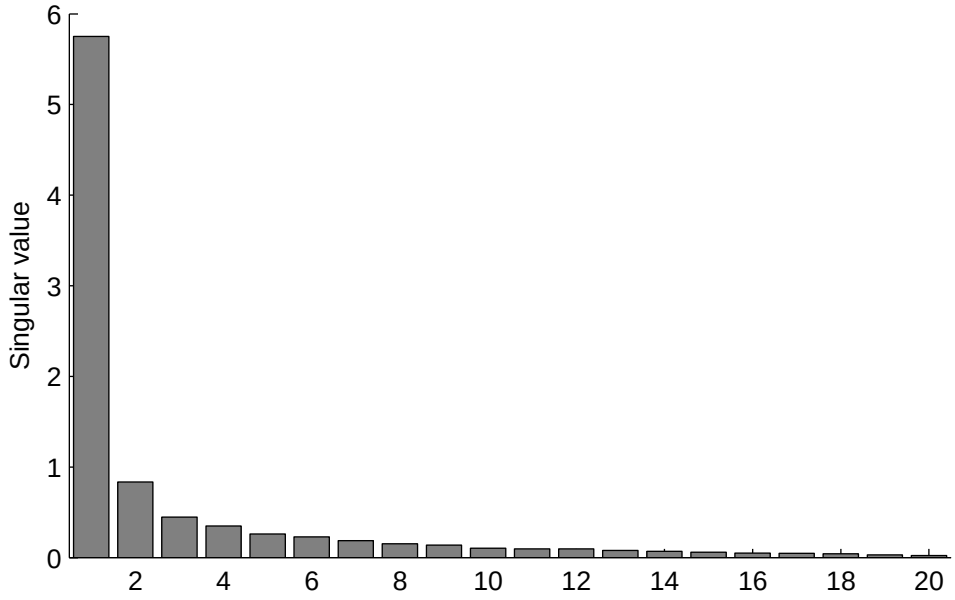

Figure 6.7: Singular values of the gradient matrix for logic circuit problem, after 20 iterations of the subspace discovery algorithm. Active subspace may be as small as 1-dimensional depending on the threshold.

parameters as the active parameters. However, depending on the threshold, as many as 100 parameters could be considered active.

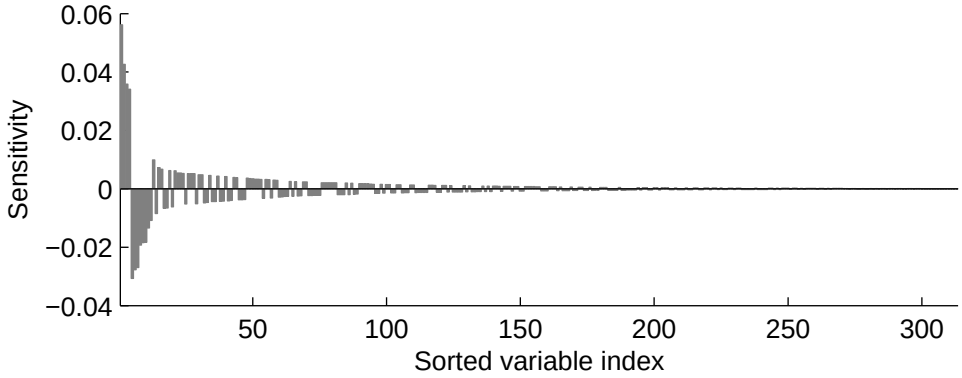

Figure 6.8: Sensitivity coefficients GRI-Mech 3.0 target CH3.C1a from a Hadamard design.

Even if the subspace discovery algorithm is performed in all 313 variables, a low dimensional active subspace is found. Figure 6.9 shows the singular values of the gradient matrix after 22 iterations of the algorithm.

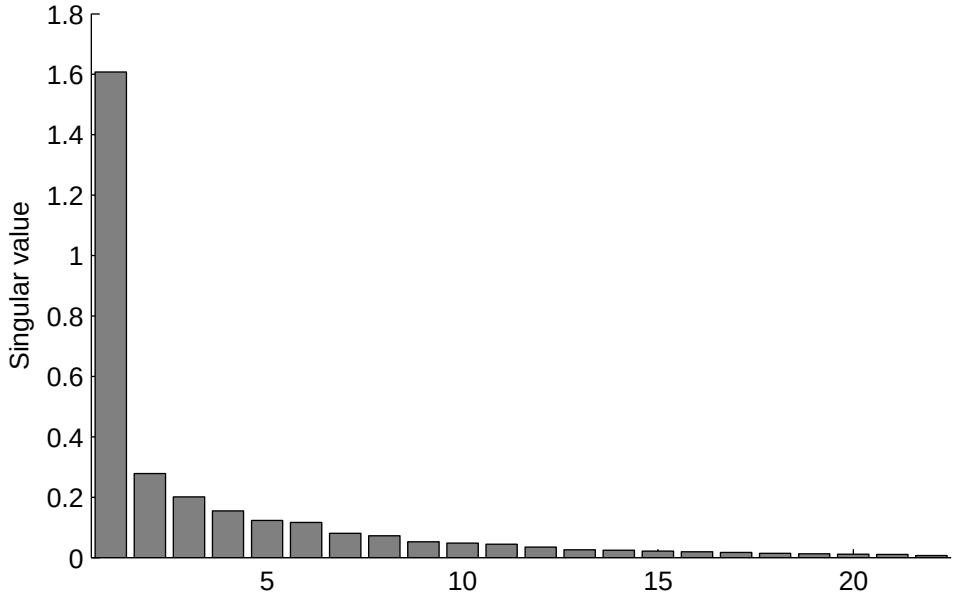

Figure 6.9: Singular values of the gradient matrix for GRI-Mech 3.0 target CH3.C1a after 22 iterations of the subspace discovery algorithm.

6.6.4 Cellular Calcium Response

The next example comes from a biological signaling application. The model simulates the calcium response to ligand application in a murine macrophage cell line [62]. We have used the model in previous work for model discrimination [32]. The model involves 8 coupled ODEs with 34 uncertain parameters and has several features that were previously examined. For this example, we examine the maximum calcium concentration response to a 100nM ligand application.

Figure 6.10 shows the active variables for the response feature. Figure 6.11 shows the singular values of the gradient matrix after 10 iterations of the active subspace discovery algorithm. It appears that the model depends mostly on a 1-dimensional subspace, and at most on a 5 dimensional subspace. Figure 6.12 shows the feature along the one dimensional subspace and how much the response surface varies from the 1- dimensional subspace approximation.

6.7 Extra Iteration Heuristic

The active subspace discovery algorithm includes an automated way of detecting the rank based on a threshold parameter (line 9 in Algorithm 6.1). The quality of this automated rank cutoff can change with the particular choice of locations for gradient computation (line 3 in Algorithm 6.1). The algorithm calls for a Latin hypercube design which has an element of randomness but provides a design that is "spread

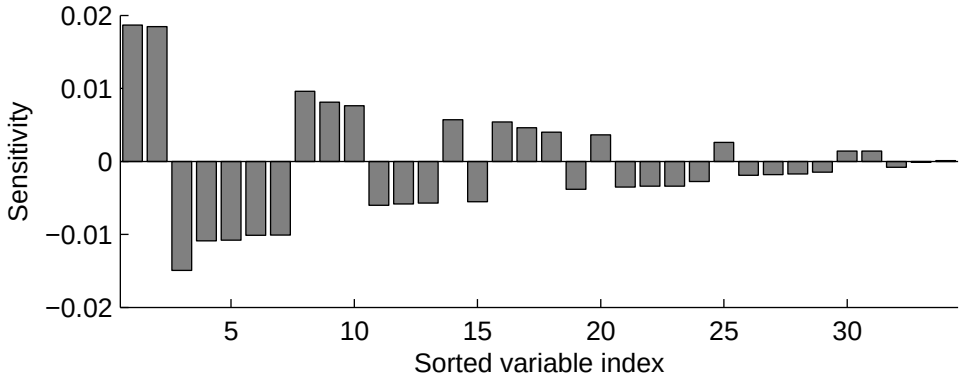

Figure 6.10: Sensitivity coefficients calcium response model from a Hadamard design.

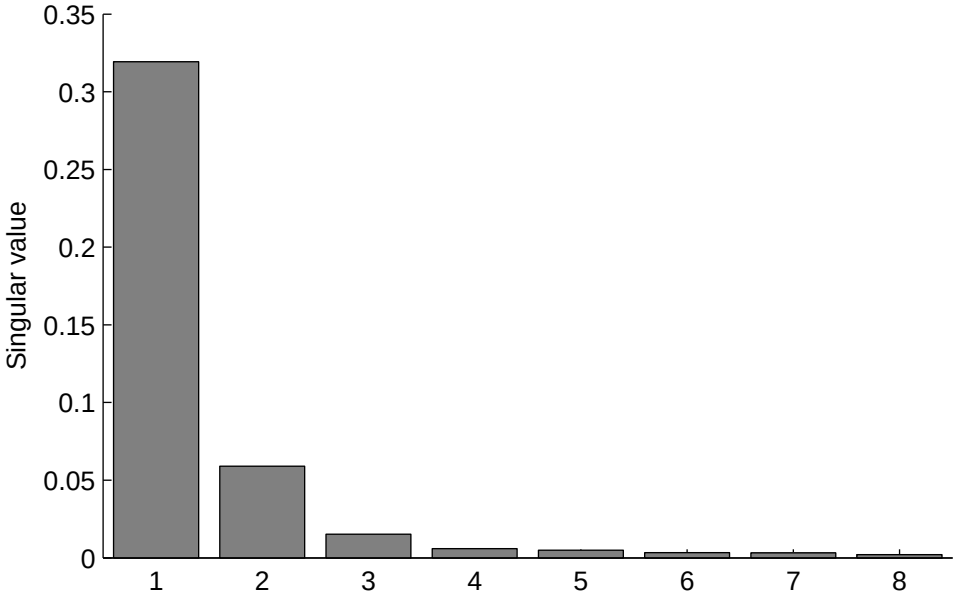

Figure 6.11: Singular values of the gradient matrix for the calcium response model after 15 iterations of the subspace discovery algorithm.

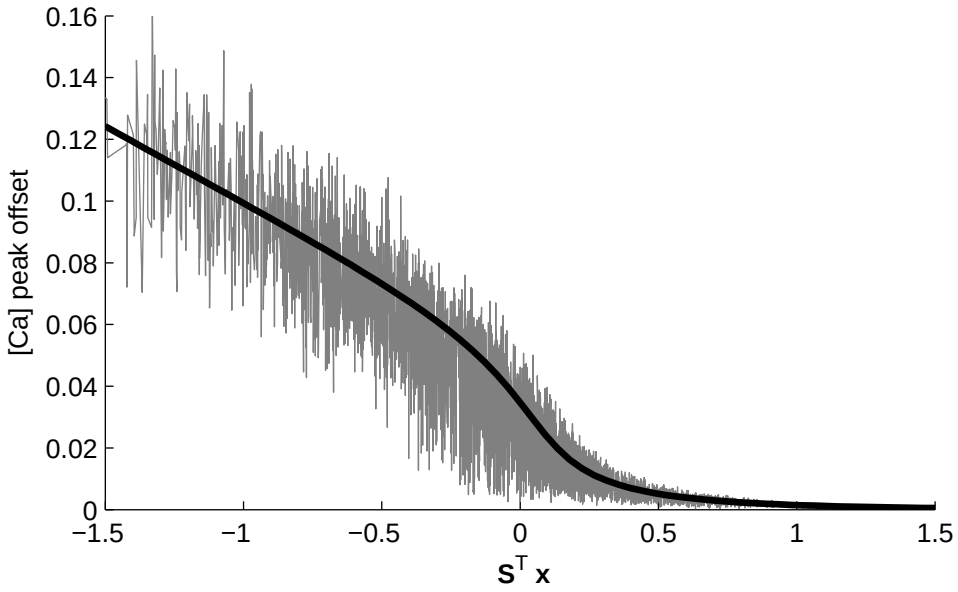

Figure 6.12: Peak calcium concentration offset from initial condition along 1-dimensional active subspace. Light gray line represents the variance in the response across all 34 parameters. The black line is the response as it varies only on the subspace.

out" in a hypercube [70]. The random element of this design means the algorithm will yield slightly different results each time.

Take, for example, the function f_1 from Section 6.6.1 with a fixed 30×5 subspace matrix $\mathbf{S}$. We ran the active subspace discovery algorithm 5,000 times with a threshold $\tau = 0.05$. 4,242 of the runs correctly computed an active subspace dimension of 5, 748 (almost 15%) computed a dimension less than 5, and the other 10 runs computed a dimension greater than 5. These results are summarized in the first row of Table 6.1. The computations that resulted in an overestimate of the active subspace are not too problematic, because they are conservative in this way, and furthermore, the subspace containment measure $m(\mathcal{R}(\mathbf{S}), \mathcal{R}(\mathbf{S}_{\text{est}}))$ for these cases is relatively good with an average value of 0.0015. However, in the case of underestimating the subspace dimension, which occurs more often than overestimating, the measure is always 1 because the active subspace cannot possibly be contained in an estimate with a smaller dimension.

The accuracy of computing the active subspace matrix should be improved with an increased number of gradients. However, more gradients means more simulations. So as a heuristic, it is suggested that one or two more gradients be calculated after the threshold is met, after which the rank is recalculated. Repeating the analysis on function f_1 (5,000 calculations of the active subspace) using either one or two more iterations of the procedure yields the results in the second and third rows of Table 6.1. The heuristic decreases the number of underestimates of the dimension of the active subspace.

As already mentioned, each extra iteration requires more gradient calculations and hence more simulation time. Even without any extra iterations the algorithm was successful about 85% of the time in the example above. The success rate is increased to almost 97% with only one extra iteration. If one extra gradient calculation is reasonable given the simulation expense, then it is recommended.

Table 6.1: Count of active subspace dimension calculations from 5000 iterations of the algorithm on function f_1 from Section 6.6.1 with a threshold of $\tau = 0.05$. The first row represents the counts from running the unmodified procedure in Algorithm 6.1. The next two rows represent the counts by including either one or two more iterations after the singular value threshold is met.

	dim< 5	dim$= 5$	dim> 5
Unmodified alg.	748	4242	10
1 extra iteration	146	4828	26
2 extra iteration	23	4953	24

6.8 Active Subspace Discovery of a Vector-Valued Function

The subspace discovery algorithm works well for functions with multiple outputs. Let the function f now be a map from $\mathbb{R}^n$ to $\mathbb{R}^m$. This is trivially thought of as m functions that map $\mathbb{R}^n$ to $\mathbb{R}$. Let f_j be the function associated with the j^{th} output of f, with $j \in \{1, \ldots, m\}$. Each f_j may have its own distinct active subspace, $\mathbf{S}_j$. Therefore, active subspace discovery and surrogate fitting must be performed for each output of f.

The procedure as written in Algorithm 6.1 can simply be repeated for each of the m outputs of f, one at a time. However, it is often the case that computation of each of f's outputs is not independent of the others. For example, the elements of the output of f may be different scalar features or attributes of a simulation output as it evolves over time. In this case, it makes sense to run each iteration of the subspace algorithm simultaneously for each output of the function. The total number of iterations depends on the largest dimension of all the outputs' active subspaces. This implies that for some outputs there will be more iterations than necessary to discover the rank of the corresponding active subspace. However, as mentioned in Section 6.7, this is beneficial.

Algorithm 6.2 shows the procedure used for a multi- output function f. In this algorithm, the Jacobian is only calculated once per location (line 6 in Algorithm 6.2). This is a significant improvement over computing the gradient for each element of the function's output if the computation of the Jacobian (i.e. the gradient for all

outputs) takes only as many simulations to compute as it does to compute one of the gradients.

Algorithm 6.2 Active Subspace Discovery for a Vector-Valued Function

Require: f {function of interest}
Require: $\mathcal{H}$ {n-dimensional hyperrectangle domain for f}
Require: τ {Singular value cutoff threshold, 0.1 or 0.05 recommended}
$n \leftarrow$ dimension of $\mathcal{H}$.
$m \leftarrow$ dimension of the output of f.
Initialize matrices $\mathbf{F}_j$ for $j = 1, \ldots, m$ (empty)
$\{\texttt{dloc}_k\}_{k=1}^n \leftarrow$ n-point, n-dimensional Latin hypercube design on $\mathcal{H}$.
for $k = 1$ to n **do**
  $\mathbf{J} \leftarrow$ Jacobian of f at location $\texttt{dloc}_k$
  for $j = 1$ to m **do**
    Add j^{th} row of $\mathbf{J}$ to bottom of matrix $\mathbf{F}_j$.
    $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} \leftarrow$ SVD of $\mathbf{F}_j$ {such that $\mathbf{F}_j = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$}
    $\boldsymbol{\sigma} \leftarrow \text{diag}(\mathbf{\Sigma})$
    if $k > 1$ **and** there exists an i such that $\sigma_{i+1} \leq \tau\sigma_i$ **then**
      $r_j \leftarrow$ the minimum i such that $\sigma_{i+1} \leq \tau\sigma_i$
      $\mathbf{S}_j \leftarrow$ first r columns of $\mathbf{V}$
    else if $k == n$ **then**
      $\mathbf{S}_j \leftarrow \mathbf{V}$
    end if
  end for
  if r_j is set for all $j = 1, \ldots, m$ **then**
    return $\mathbf{S}_j$ for $j = 1, \ldots, m$
  end if
end for

6.9 Alternative Approach using Rank Minimization

The methodology for active subspace discovery presented in Section 6.2 appears to work fairly well. In the two toy examples where the active subspace is known (Section 6.6.1), the subspace containment measure was very low, implying a good agreement between the true active subspace and the approximation.

One of the biggest flaws with the procedure occurs when gradients are not available from the simulations, and numerical gradients are instead calculated. In this case, many function evaluations must occur in tight bunches to estimate the gradients. In Chapter 7 function evaluations will be used to make surrogate fits in the subspace

dimension. Only one evaluation for each bunching of points used to discover the active subspace can be reused in making the surrogate fit. This is because a good experiment design for regression has high variance, i.e. is made of points that are spread out.

This section presents an attempted approach at discovering the active subspace using only partial gradients, or gradients along certain directions. If only partial gradients are calculated, less points are needed in each bunch, and a larger percentage of the evaluations can be used again for surrogate fitting. The method presented does not appear to work very well; however, we describe the method and present an example for the sake of thoroughness.

As before, define functions $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^r \to \mathbb{R}$, and a matrix $\mathbf{S} \in \mathbb{R}^{n\times r}$ where $\mathbf{S}^\mathsf{T}\mathbf{S} = \mathbf{I}$ and suppose that the relation $f(\mathbf{x}) = g(\mathbf{S}^\mathsf{T}\mathbf{x})$ holds. Say we compute the gradient of f at a point $\mathbf{x}_k$ along the m directions defined by the linearly independent columns of $\mathbf{V}_k \in \mathbb{R}^{n\times m}$. Then the resulting $1 \times m$ directional gradient vector $\mathbf{d}_k$ is

$$\mathbf{d}_k = \nabla f(\mathbf{x}_k) \cdot \mathbf{V}_k. \tag{6.17}$$

Here the assumption is that $\nabla f(\mathbf{x}_k)$ is not computed, only $\mathbf{d}_k$ is. For a numerical gradient this would only require $m + 1$ function evaluations near $\mathbf{x}_k$. Given that f only varies along the active subspace $\mathcal{R}(\mathbf{S})$, equation (6.17) is further decomposed as

$$\mathbf{d}_k = \nabla g(\mathbf{S}^\mathsf{T}\mathbf{x}_k) \cdot \mathbf{S}^\mathsf{T} \cdot \mathbf{V}_k. \tag{6.18}$$

Using previous notation, $\mathbf{F}$ is the matrix of gradients of f at the points $\{\mathbf{x}_k\}_{k=1}^N$ and $\mathbf{G}$ is the matrix of gradients of g at the points $\{\mathbf{S}^\mathsf{T}\mathbf{x}_k\}_{k=1}^N$. The directional gradient can be written using these matrices as

$$\mathbf{d}_k = \mathbf{e}_k^\mathsf{T}\mathbf{F}\mathbf{v}_k = \mathbf{e}_k\mathbf{G}\mathbf{S}^\mathsf{T}\mathbf{V}_k, \tag{6.19}$$

where $\mathbf{e}_k$ is the k^{th} N-dimensional standard basis vector.

In this set-up, the matrix $\mathbf{F}$ is unknown. However, we know it factors into the matrix product $\mathbf{G}\mathbf{S}^\mathsf{T}$ where the inner dimension is r, a (hopefully) small number. Therefore, the problem becomes finding a low rank matrix $\mathbf{F}$ such that all the directional derivatives match the calculations. Formulated as an optimization problem, this is written as

$$\begin{aligned} &\min_{\mathbf{F}} \ \operatorname{rank}(\mathbf{F}) \\ &\text{s.t. } d_k = \mathbf{e}_k^\mathsf{T}\mathbf{F}\mathbf{V}_k, \ k = 1, \ldots, N. \end{aligned} \tag{6.20}$$

This is a difficult problem to solve. However, over the domain of matrices in the unit ball, the convex envelope of the rank function is the sum of the singular values which is called the nuclear norm and notated $\|\cdot\|_*$ [31]. Even when $\mathbf{F}$ is not in the unit ball, the nuclear norm provides a good heuristic approximation.

$$\begin{aligned} &\min_{\mathbf{F}} \ \|\mathbf{F}\|_* \\ &\text{s.t. } \mathbf{d}_k = \mathbf{e}_k^\mathsf{T}\mathbf{F}\mathbf{V}_k, \ k = 1, \ldots, N. \end{aligned} \tag{6.21}$$

This is a convex problem, and furthermore, Benjamin Recht it et. al. [89] showed that this can be recast as the semidefinite program

$$\begin{aligned} \min_{\mathbf{F},\mathbf{W}_1,\mathbf{W}_2} \ & \frac{1}{2}(\mathrm{Tr}\left[\mathbf{W}_1\right] + \mathrm{Tr}\left[\mathbf{W}_2\right]) \\ \text{s.t.} \ & \begin{cases} \begin{bmatrix} \mathbf{W}_1 & \mathbf{F} \\ \mathbf{F}^\intercal & \mathbf{W}_2 \end{bmatrix} \succeq 0 \\ \mathbf{d}_k = \mathbf{e}_k^\intercal \mathbf{F} \mathbf{V}_k, \ k = 1, \ldots, N. \end{cases} \end{aligned} \tag{6.22}$$

The number of constraints in the nuclear norm formulation (equation (6.21)) is $N \cdot m$. The matrix $\mathbf{F}$ has $N \cdot n$ elements in it. Both N and m need to be greater than the underlying r. Since r is unknown *a priori*, it makes sense to set N to be a large number, and increase m until the rank stops increasing. As N increases, so too does the difference between the number of entries of $\mathbf{F}$ and the number of constraints. Therefore, setting $N = n$ appears to make the most sense.

This implies an algorithm that is similar to Algorithm 6.1. The algorithm cycles through the N locations adding one new derivative direction to the current location. The SDP in equation (6.22) is solved yielding the matrix $\mathbf{F}$. From here the procedure continues as before; evaluating the singular values of $\mathbf{F}$ to determine the dimension of the active subspace.

As an example application, let f be defined as the first toy example in Section 6.6.1, equation (6.15). The function has 30 variables, and depends on a 5 dimensional space. Figure 6.13 shows the active subspace dimension estimate as the algorithm iterates, as well as the corresponding subspace containment measure $m(\mathcal{R}(\mathbf{S}), \mathcal{R}(\mathbf{S}_{\text{est}}))$. It appears that the rank estimate settles to be either 5 or 6 when there are around twelve derivative directions per location. However, the subspace containment measure shows that the active subspace estimate is very poor at this point. It is not until there are 20 or 25 derivative directions per location that the active subspace estimate is relatively good. In general cases, however, since there is no way to evaluate the subspace containment measure (because the actual active subspace is unknown), there is no way to know when the iteration should stop. Moreover, 25 derivative directions yields a gradient that is almost complete (the full gradient has 30 directions).

This methodology of rank minimization does not appear to work for this application. Several combinations of choices of N and m yielded similar or worse results. Therefore, this method is not recommended.

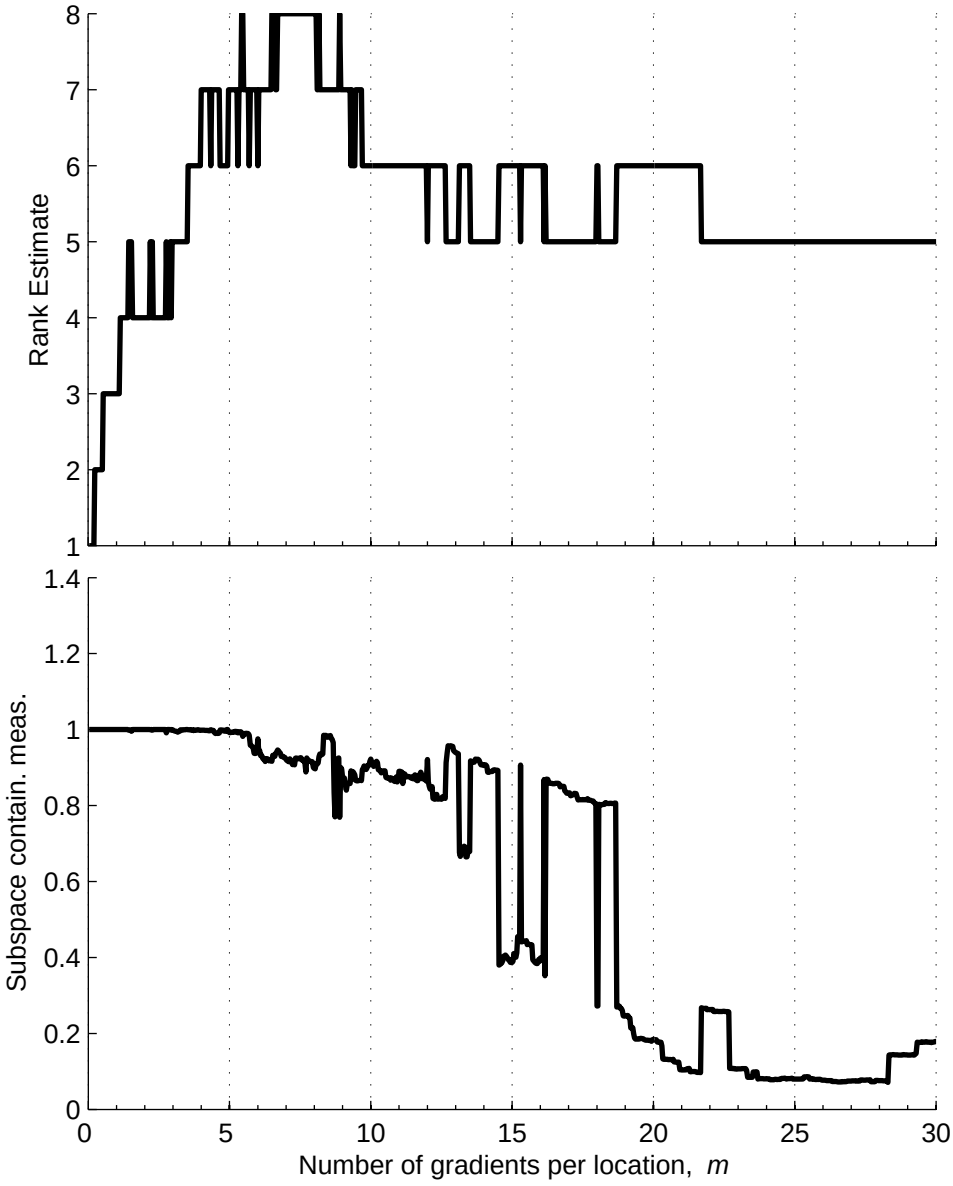

Figure 6.13: Rank estimate and subspace containment measure compared to iteration of an active subspace discovery algorithm using directional derivatives. The rank is more or less found well before the estimate of the active subspace is a good approximation. The horizontal scale is marked with the minimum number of direction derivatives at each location.

Chapter 7

Subspace Experiment Design for Fitting Surrogate Models

The design of experiments (DoE) is the process by which a set information- gathering processes are formulated. In the case of both physical and computer experiments, DoE often refers to the process of selecting specific sets of parameter values at which the experiment will be performed. The goal of the design is to gather information about a system of interest by adequately sampling the experiment parameter space within the physical constraints and practical considerations of the experiment [13].

The goal of the present study is to produce a technique of creating surrogate models based on an active subspace. Take a function f with n input parameters and a scalar output. Typical surrogate modeling techniques involve evaluating f at a set of points $\{\mathbf{x}_j\}_{j=1}^m$, and fitting a function of the desired form to this evaluation data [70]. In this context, experiment design is the task of choosing the set of points for evaluation.

Most classical experiment designs were created for hypercube and spherical domains. These include grid (factor) designs, Latin squares, Box-Behnken designs, and equiradial designs to name only a few [13, 75, 118, 122]. Some work has also been done to create designs in very high dimensions [5, 80], but these also focus on hypercube domains. One notable exception arises in mixture experiments, where the values of each parameter must add to a constant number. The domain for mixtures is a simplex, over which several classes of design exist [75].

In Chapter 6, we explored an algorithm for discovering an active subspace of a function's parameters. This subspace can be represented as a transformation of the parameters to a lower dimensional space. The goal is to make a surrogate fit that is a function of the transformed variables. Therefore an experiment design of parameters

for the surrogate fit is in the transformed space. The domain of the parameters in the full (original) coordinates, is a hyperrectangle $\mathcal{H}$. When the hyperrectangle constraints are projected onto the lower dimensional subspace, the result is a polytope.[1] The goal of this chapter is to create good experiment designs over such polytopes, which can be transformed back to the original variables for simulation, evaluation, or experimentation.

Designs created for hyperrectangle or ellipsoid domains cannot be used with a polytope domain. Simplexes are a special-case of polytopes, and therefore it is not surprising that some simplex designs can work for general polytopes. Simplex-lattice designs include each simplex vertex and various combinations of points evenly spread along the edges and faces. Simplex-centroid designs also include the center of the simplex. It seems logical, that this type of design would work well for a general polytope. However, the problem of finding all the vertices of a polytope is NP-hard [54]. Furthermore, the number of vertices can be much larger than the desired size of the design. The design method presented in this chapter creates a reasonably sized design, and does not require that the vertices of the polytope are known.

7.1 Methodology Overview

The goal of experiment design in this case is to choose the parameter vectors where evaluations will make a good surrogate fit. Suppose that the function of interest f at a point $\mathbf{x}_j$ outputs the noisy evaluation y_j and has the form

$$y_j = \sum_{\ell=1}^{L} a_\ell \phi_\ell(\mathbf{x}_j) + \eta_j \tag{7.1}$$

where the functions $\phi_\ell : \mathbb{R}^n \to \mathbb{R}$ are basis functions and η_j is measurement noise. The set of noises $\{\eta_j\}_{j=1}^m$ is modeled as independent Gaussian random variables with zero mean and unit variance. Using a vector notation, let $\mathbf{a} \in \mathbb{R}^L$ be the vector of a_ℓ's and $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^L$ be the map whose outputs correspond to each map ϕ_ℓ. Then $y_j = \mathbf{a}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_j)$ for each $j = 1, \ldots, m$. Also, assume that there is enough data to determine the coefficients $\mathbf{a}$, in other words, the vectors $\boldsymbol{\phi}(\mathbf{x}_j)$ span $\mathbb{R}^m$.

The least squares estimate of the coefficient vector is

$$\hat{\mathbf{a}} = \left(\sum_{j=1}^{m} \boldsymbol{\phi}(\mathbf{x}_j)\boldsymbol{\phi}^\mathsf{T}(\mathbf{x}_j) \right)^{-1} \sum_{j=1}^{m} y_j \boldsymbol{\phi}(\mathbf{x}_j). \tag{7.2}$$

[1] Definitions of polytopes vary. Here we use one common definition as in [20]: A *polyhedra* is the intersection of a finite number of half-spaces (those defined by a linear inequality). A *polytope* is a closed polyhedra.

This is a random vector due to y_j's dependence on η_j. The error in the coefficient vector $\mathbf{e} = \hat{\mathbf{a}} - \mathbf{a}$ is a random vector with zero mean and covariance matrix

$$\boldsymbol{\Sigma} = \mathrm{E}\left[\mathbf{e}\mathbf{e}^{\mathsf{T}}\right] = \left(\sum_{j=1}^{m} \boldsymbol{\phi}(\mathbf{x}_j)\boldsymbol{\phi}^{\mathsf{T}}(\mathbf{x}_j)\right)^{-1}. \tag{7.3}$$

The objective of experiment design in this context is to choose the vectors $\{\mathbf{x}_j\}_{j=1}^m$ such that the error has a "small" covariance matrix $\boldsymbol{\Sigma}$. The scale of the covariance matrix can be determined using some scalarization such as the determinant, largest singular value, or trace of the matrix. This objective is equivalent to designing a set of points whose variance, $\boldsymbol{\Sigma}^{-1}$, is large. There are several scalarizations that are used to determine how to compare designs, which are discussed further in Section 7.3.

Boyd and Vandenberghe use a similar setup and go on to present a special type of experiment design where the goal is to pick the best design points from a larger set of predetermined parameter vectors [14§7.5]. In order to create designs in a polytope, our methodology is to first create a large sample of points in the polytope and then use Boyd and Vandenberghe's method of selecting a design from this sample. Section 7.2 will discuss a couple of different ways of creating large samples in a polytope and Section 7.3 will discuss several optimizations for choosing experiment designs.

7.2 Sampling the Constrained Subspace

Let the function $f : \mathbb{R}^n \to \mathbb{R}$ vary solely in directions along an r- dimensional active subspace spanned by the orthonormal columns of the $n \times r$ matrix $\mathbf{S}$. Then f can be decomposed as

$$f(\mathbf{x}) = g(\mathbf{S}^{\mathsf{T}}\mathbf{x}) \tag{7.4}$$

where g is a function of only r variables. The domain of f is a hyperrectangle $\mathcal{H} \subseteq \mathbb{R}^n$. Let the input of the function g be denoted as $\mathbf{z} = \mathbf{S}^{\mathsf{T}}\mathbf{x}$, the vector of transformed variables.

The experiment design for surrogate model fitting should be in the transformed coordinates: the $\mathbf{z}$-space. The following subsections demonstrate methods for sampling a set of vectors $\{\mathbf{z}_k\}_{k=1}^N$ such that each vector $\mathbf{z}_k$ corresponds to a vector $\mathbf{x}_k \in \mathcal{H}$.

7.2.1 Uniform Sample of a Polytope

The intersection of the subspace $\mathcal{R}(\mathbf{S})$ and the domain $\mathcal{H}$ is a polytope. To guarantee a nonempty polytope, assume that the parameters $\mathbf{x}$ have been translated

such that $\mathcal{H}$ is centered at the origin.[2] For every point $\mathbf{x} \in \mathcal{R}(\mathbf{S})$, there exists a vector $\mathbf{z} \in \mathbb{R}^r$ such that $\mathbf{x} = \mathbf{S}\mathbf{z}$. If $\mathbf{x}$ is constrained to the hyperrectangle $\mathcal{H}$, then this imposes constraints on the vector $\mathbf{z}$. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be the vectors of lower and upper bound constraints defining $\mathcal{H}$. Then

$$\boldsymbol{\alpha} \leq \mathbf{S}\mathbf{z} \leq \boldsymbol{\beta} \tag{7.5}$$

is a system of linear constraints on $\mathbf{z}$. Therefore $\mathbf{z}$ is constrained to a polytope.

Traditional sampling techniques such as sampling from uniform or Gaussian distributions in r dimensions do not work for sampling this polytope as they do not take into account constraints that are not coordinate aligned. A uniform distribution can exist over a non-hyperrectangle domain, but it is not necessarily easy to sample or even describe. In this section, we describe an algorithm designed to sample a polytope.

The concept has been written about in the context of random walks [115] and a Gas Kinetics Point Generation Algorithm (GKPG) [60]. The algorithm described here takes the latter context. The concept begins by filling the polytope with a hypothetical gas. The algorithm tracks a single gas particle (in n dimensions) as it makes collisions with other particles and with set boundaries given a very simple dynamic model.

Start with a point in the polytope. Pick a velocity from a Gaussian distribution (which is uniform in direction [66, 73]) in r dimensions. Allow the particle to travel with the speed and direction described by the velocity vector. If the particle encounters a set boundary, it reflects off of the boundary. After one time unit, the particle's position is recorded as a sample point. A new velocity is chosen simulating a collision with another particle. The process is then repeated until the desired number of samples is obtained. Algorithm 7.1 shows the steps in the procedural form.

Several things should be noted in Algorithm 7.1. Since the polytope is convex, the sample will converge to a uniform sampling of the polytope as the number of samples goes to infinity [115]. Furthermore, the variance of the velocity (line 4) and the time between recorded samples (line 5) can affect the time the algorithm takes to achieve a satisfactory spread of points. If either the velocity variance and time between recorded samples is too large, i.e. the distance traveled by the particle is too large, there may be lots of reflections which increase the computation time to discover the constraint involved and location of the collision. If distance traveled between samples is too small, it may take a long time to sufficiently cover the space. To combat the problem of too many reflections, Algorithm 7.1 limits the number of reflections to 6 (line 16).

The algorithm heuristically chooses the variance of the velocity vector to depend on the Chebyshev radius of the polytope (line 2 in Algorithm 7.1). The Chebyshev

[2]It is possible that the intersection of the subspace $\mathcal{R}(\mathbf{S})$ and $\mathcal{H}$ is empty. Translating the parameters such that $\mathcal{H}$ is centered at the origin is sufficient (but not necessary) to achieve a nonempty polytope. This translation does not affect the subspace dependence of the function f.

Algorithm 7.1 Gas Dynamics Sampling Algorithm for Polytopes

Require: N {Number of desired samples}
Require: $\{\mathbf{a}_i\}_{i=1}^p$, $\{b_i\}_{i=1}^p$ {Vectors and scalars defining constraints $\mathbf{a}_i^\mathsf{T}\mathbf{z} \leq b_i$.}
$\mathbf{z}(0) \leftarrow$ Chebyshev center of the polytope $\{\mathbf{z} \ : \ \mathbf{a}_i^\mathsf{T}\mathbf{z} \leq b_i, \ i = 1, \ldots, p\}$.
$\texttt{radius} \leftarrow$ Chebyshev radius of the polytope.
for $k = 1$ to N **do**
  Choose $\mathbf{v} \in \mathbb{R}^r$ from a Gaussian distribution $\mathcal{N}(\mathbf{0}, \texttt{radius}^2/9)$ {"velocity"}
  $T_{\text{remain}} \leftarrow 1$ {Remaining "time" to next sample}
  $\mathbf{z}_{\text{now}} \leftarrow \mathbf{z}(k-1)$ {Current particle position}
  cnt $\leftarrow 0$ {Number of reflections for this point}
  while $T_{\text{remain}} > 0$ **do**
    $\{T^\star, i^\star\} \leftarrow \min_i \max_T T$ s.t. $\mathbf{a}_i^\mathsf{T}(\mathbf{z}_{\text{now}} + T\mathbf{v}) \leq b_i$.
    if $T^\star > T_{\text{remain}}$ {Didn't hit boundary} **then**
      $\mathbf{z}(k) \leftarrow \mathbf{z}_{\text{now}} + T_{\text{remain}}\mathbf{v}$
      $T_{\text{remain}} \leftarrow 0$
    else if $T^\star \geq 0.99T_{\text{remain}}$ {Near boundary} **then**
      $\mathbf{z}(k) \leftarrow \mathbf{z}_{\text{now}} + 0.99T_{\text{remain}}\mathbf{v}$
      $T_{\text{remain}} \leftarrow 0$
    else if cnt$= 6$ **then**
      $\mathbf{z}(k) \leftarrow \mathbf{z}_{\text{now}} + 0.99T^\star\mathbf{v}$
      $T_{\text{remain}} \leftarrow 0$
    else
      $\mathbf{z}_{\text{now}} \leftarrow \mathbf{z}_{\text{now}} + 0.999T^\star\mathbf{v}$
      $T_{\text{remain}} \leftarrow T_{\text{remain}} - T^\star$
      $\boldsymbol{\eta} \leftarrow \mathbf{a}_{i^\star}$ {Boundary normal vector}
      $\mathbf{v} \leftarrow \mathbf{v} - 2\frac{\boldsymbol{\eta}^\mathsf{T}\mathbf{v}}{\boldsymbol{\eta}^\mathsf{T}\boldsymbol{\eta}}\boldsymbol{\eta}$ {Reflect velocity vector}
    end if
    cnt $\leftarrow$cnt+1
  end while
end for
return $\{\mathbf{z}(k)\}_{k=0}^N$

radius is the radius of the largest spheroid that is a subset of the polytope. The Chebyshev center is the center of this spheroid, and the sampling algorithm uses this as the initial sample point. The Chebyshev radius and center are easily computed for polytopes using the linear program [14],

$$\begin{aligned} \{\mathbf{c}^\star, r^\star\} = &\underset{\mathbf{c},r}{\operatorname{argmax}}\, r \\ &\text{s.t. } \mathbf{a}_i^\mathsf{T}\mathbf{c} + r\|\mathbf{a}_i\|_2 \leq b_i,\ i = 1, \ldots, p. \end{aligned} \tag{7.6}$$

The Chebyshev center is not unique in general, but the Chebyshev radius is.

When the hypothetical gas particle is traveling in a given direction, computing the first intersected boundary is simple. The optimization on line 9 of Algorithm 7.1 does not require an optimization solver. For each i, the maximum T is $(b_i - \mathbf{a}_i^\mathsf{T}\mathbf{z}_{\text{now}})/(\mathbf{a}_i^\mathsf{T}\mathbf{v})$ if $\mathbf{a}_i^\mathsf{T}\mathbf{v}$ is positive, and is infinite otherwise. Finding the minimum i corresponds to finding the minimum entry in a list.

This algorithm is efficient and can generate a large number of samples very quickly. A simple example in two dimensions is shown in Figure 7.1. Each point in the sample, $\mathbf{z}_k$, is associated with a point $\mathbf{x}_k = \mathbf{S}\mathbf{z}_k$ in $\mathbb{R}^n$ that is contained in $\mathcal{H}$, due to the constraints imposed by the sampling algorithm. Note that if any translations were performed on the parameters prior to the polytope sampling step, a simple reverse translation of variables is needed here to achieve valid $\mathbf{x}$'s.

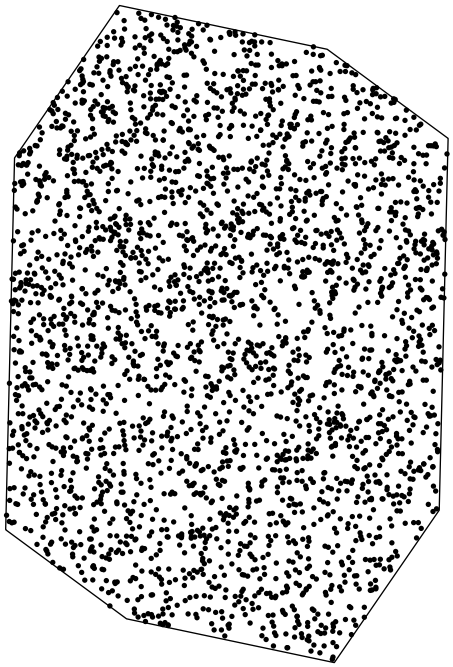

Figure 7.1: Sample of a 2-dimensional polytope using the Gas Dynamics Sampling Algorithm.

7.2.2 Sampling a Polytope Via Point Projection

The second method of sampling is much simpler conceptually. Let $\{\mathbf{x}_k\}_{k=1}^N$ be a sample of points on the n-dimensional hyperrectangle $\mathcal{H}$. The corresponding sample in the transformed variables is simply $\{\mathbf{z}_k = \mathbf{S}^\mathsf{T}\mathbf{x}_k\}_{k=1}^N$. The domain of possible vectors $\mathbf{z}_k$ is the transformation of $\mathcal{H}$ through $\mathbf{S}^\mathsf{T}$, which is a polytope. Note that this sampling method does not work for arbitrary polytopes, but only ones that are projections of higher dimensional hyperrectangles.

The initial sample of $\mathcal{H}$ can be obtained using any sampling method that is efficient and produces many points. A simple choice would be a uniform sample over $\mathcal{H}$. It is important to note, however, that a uniform sample on $\mathcal{H}$ will not result in a uniform sample over the projected polytope. Furthermore, when projecting from a high dimension down to a lower one, the corners of the hyperrectangle are not sampled well (because most of the volume is near the edges) and therefore the projected points are not near the boundaries of the polytope. Take Figure 7.2 as an example. The line in the figure represents the boundary of the polytope formed by projecting the hypercube $[-1,\ 1]^{30}$ onto an arbitrary 2-dimensional subspace. The points are a uniform sample in the hypercube.

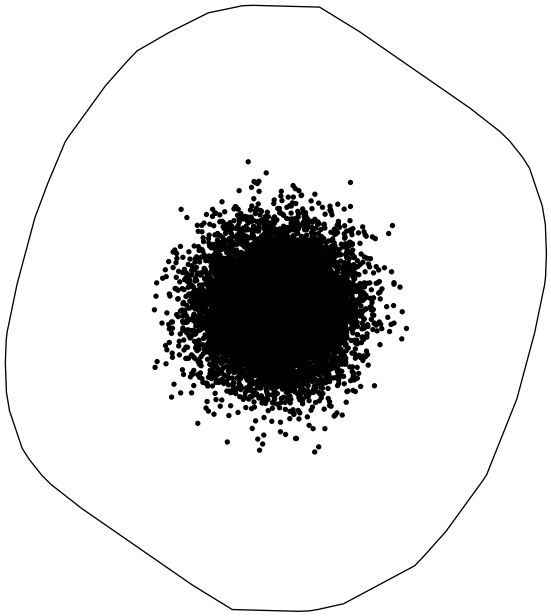

Figure 7.2: A 10000-point uniform sample of points over the hypercube $[-1,\ 1]^{30}$ projected onto an arbitrary 2-dimensional subspace.

A slightly better approach when n is fairly large, is to project down a sample of points taken from the vertices of the hyperrectangle. For example, in 30 dimensions, there are $2^{30} \approx 1.07 \times 10^9$ vertices. Figure 7.3 shows an example of this projection using the sample 2-dimensional subspace as the previous example. This sample does a better job of filling the polytope, but still does not fill it completely.

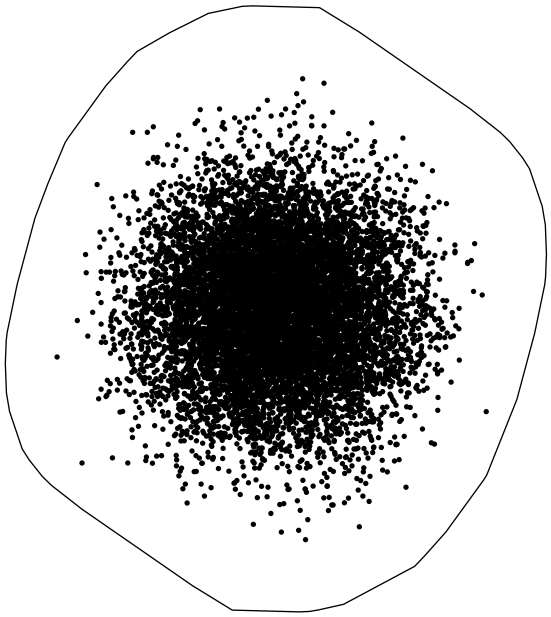

Figure 7.3: 10000 randomly chosen vertices of the hypercube $[-1,\ 1]^{30}$ project onto an arbitrary 2-dimensional subspace.

The polytope formed by projecting $\mathcal{H}$ onto the subspace $\mathcal{R}(\mathbf{S})$ is not the same as the polytope formed by intersecting $\mathcal{H}$ with $\mathcal{R}(\mathbf{S})$. The latter is the subspace used in Section 7.2.1 with the gas dynamics point generation algorithm and is easily formulated as a set of linear constraints ($\mathcal{H}$-polytope form). The polytope discussed in this section is not easily formulated in $\mathcal{H}$-polytope form. This polytope is the convex hull of all vertices of $\mathcal{H}$ projected onto the subspace. Since the number of vertices is exponential in n, this is a difficult process. The next section will further explore the differences between these two samples.

7.2.3 Comparison of Sampling Methods

The previous two subsections described methods for sampling the constrained subspace. Let Method 1 denote the method described in Section 7.2 using the gas dynamics point generation algorithm to sample the intersection of the subspace and the hyperrectangle $\mathcal{H}$. Let $\mathcal{P}_1$ denote this polytope. Let Method 2 denote the method described in Section 7.2.2 whereby a sample of the n- dimensional hyperrectangle is projected onto the subspace. Let $\mathcal{P}_2$ denote the polytope that is the projection of $\mathcal{H}$ onto the subspace.

The polytope $\mathcal{P}_2$ is almost always larger than $\mathcal{P}_1$ and in fact $\mathcal{P}_1 \subseteq \mathcal{P}_2$. Figure 7.4 shows this containment for the projection of a 2-dimensional square onto a 1-dimensional subspace. This disparity between the two polytopes is even more pro-

nounced when larger dimensions are involved. Take for example Figure 7.5, which shows the polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$ resulting from a 30-dimensional hyperrectangle and a 2-dimensional subspace. To get a sense of how much bigger $\mathcal{P}_2$ is than $\mathcal{P}_1$, examine the space $[-1,\ 1]^{30}$ and a 1-dimensional subspace. In this case, both $\mathcal{P}_1$ and $\mathcal{P}_2$ are intervals and the average interval length of $\mathcal{P}_1$ with all possible 1-dimensional subspaces is about 2.38. The average interval length of $\mathcal{P}_2$ in this case is about 4.40. These numbers appear to grow with the square root of the dimension of the hypercube, n.

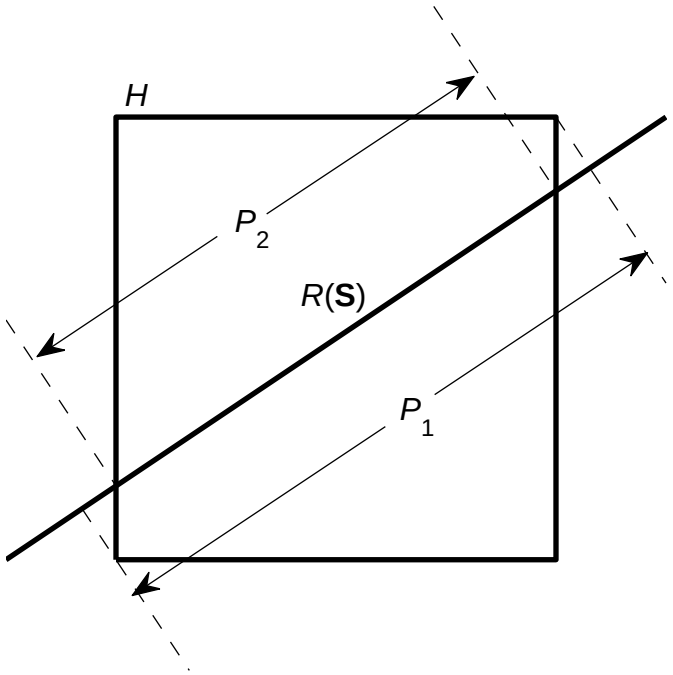

Figure 7.4: Simple example showing the difference between the two polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$ as created by taking a 2D cube and either projecting it onto or intersecting it with a 1D subspace.

Method 1 does a good job of filling $\mathcal{P}_1$ with points. Method 2 does not fill $\mathcal{P}_2$ very well (as seen in §7.2.2) but it does provide samples outside of $\mathcal{P}_1$. This is beneficial, because the ultimate goal is to have a sample with a large variance, as mentioned in Section 7.1. In this sense, Method 2 seems to be beneficial.

One problem with Method 2 is that points evenly distributed in the n-dimensional hyperrectangle tend to be clustered in the center of $\mathcal{P}_2$ after projection. Figure 7.6 shows two histograms of samples created using Method 1 and Method 2. Again, each sample is on a 2-dimensional subspace and associated with points in a 30-dimensional hypercube. In particular, the sample using Method 2 was generated using a set of the hypercube's vertices. Method 1 provides a more uniform distribution of points, while the sample from Method 2 is bunched mostly in a space tighter than that of Method 1. However, it turns out this will not affect the selection of design points.

In both methods, each sample point $\mathbf{z}_k$ has an associated point in the original n-dimensional space $\mathbf{x}_k$ that will be used to evaluate the function f. Each of the points

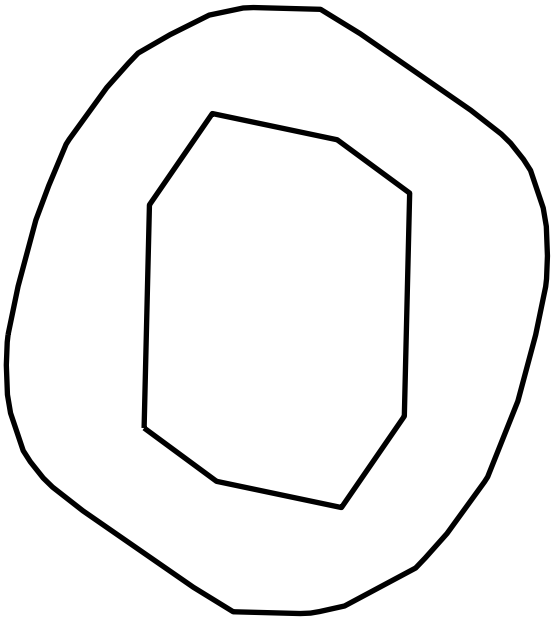

Figure 7.5: Two types of polytopes associated with the polytope sampling methodologies. The inner polytope, $\mathcal{P}_1$, is the intersection of $[-1,\ 1]^{30}$ and a 2-dimensional subspace. The outer polytope, $\mathcal{P}_2$, is the projection of $[-1,\ 1]^{30}$ onto the same subspace.

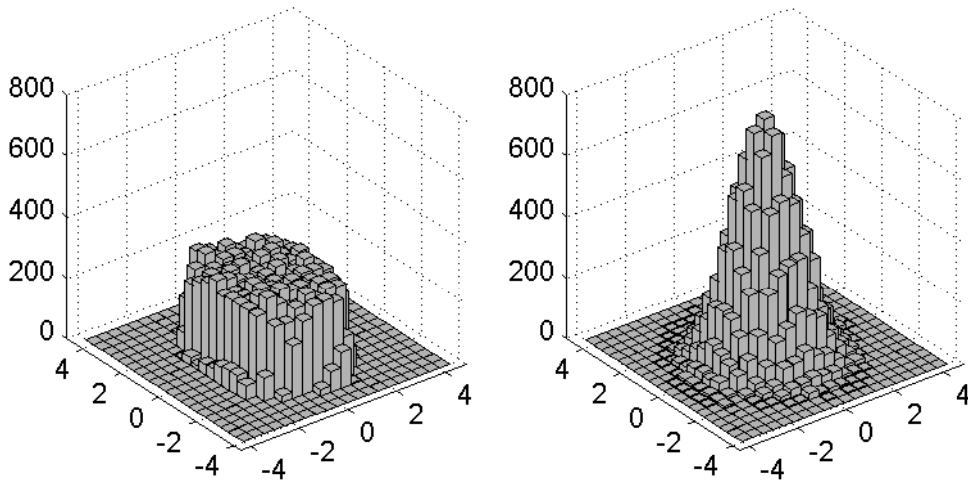

Figure 7.6: Histograms of points generated from the two polytope sampling methods. The sample for the left panel was generated with Method 1 and the sample for the right panel was generated with Method 2.

$\mathbf{x}_k$ associated with Method 1 will lie on the active subspace, whereas with Method 2 the points are may come from anywhere in the hyperrectangle $\mathcal{H}$. In this manner, Method 2 is superior. If the function f doesn't depend solely on the subspace, or the active subspace approximation has some error, it is beneficial to evaluate points off of the subspace.

Method 2 is more efficient and it appears that it will lead to better designs. Section 7.5 will provide examples to compare the two methods using surrogate fitting errors.

7.3 Experiment Design via Optimization

Section 7.2 provided a couple of methods for creating a large sample of points $\{\mathbf{z}_k\}_{k=1}^N$ in r-dimensions. The goal of this section is to choose a subset of this sample that is a good experiment design. In Section 7.1, we presented a motivation for choosing a design with a large variance, which is derived from the desire to have a low variance in the fitting error.

Specifically, the points should be spread out in the directions of the basis functions used for regression. Let $\boldsymbol{\phi} : \mathbb{R}^r \to \mathbb{R}^L$ map each sample vector to a vector in the regression basis. For example, if $r = 2$ and the desired surrogate fit has a quadratic form, then $\boldsymbol{\phi}(\mathbf{z}) = [1,\ z_1,\ z_2,\ z_1^2,\ z_1 z_2,\ z_2^2]^\mathsf{T}$.

Boyd and Vandenberghe [14] present this objective as an optimization problem which is presented here, with only slight modification.

$$\min_{\mathbf{b}} \ (\text{w.r.t. } \mathbb{S}_+^r) \ \left(\sum_{k=1}^N b_k \boldsymbol{\phi}(\mathbf{z}_k) \boldsymbol{\phi}^\mathsf{T}(\mathbf{z}_k) \right)^{-1} \qquad \text{s.t.} \ \begin{cases} b_k \in \{0,1\},\ k = 1, \ldots, N, \\ \displaystyle\sum_{k=1}^N b_k = m. \end{cases} \tag{7.7}$$

Here the minimization is over the symmetric cone of positive semidefinite matrices $\mathbb{S}_+^r$. The variables b_k are constrained to be binary, with m of them constrained to be 1. This formulation searches for the subset with m samples that provides the "smallest" error covariance matrix. This is a difficult problem to solve because of the binary variables and the nonscalar objective function.

The simplest way to address the binary variables is to relax (i.e. ignore) the binary constraint. Boyd and Vandenberghe formulate the experiment design problem with the possibility of repeated runs. This often does not work for computer simulations. However, we use their relaxation, and adjust the results to obtain a design without repetition. Define positive weights w_k that will replace the binary variables. The

relaxation simply replaces the objective with a weighted sum of all vectors [14].

$$\min_{\boldsymbol{\omega}} \ (\text{w.r.t. } \mathbb{S}_+^r) \ \left(\sum_{k=1}^{N} w_k \boldsymbol{\phi}(\mathbf{z}_k)\boldsymbol{\phi}^{\mathsf{T}}(\mathbf{z}_k)\right)^{-1} \\ \text{s.t.} \ \begin{cases} w_k \geq 0, \ k = 1, \ldots, N, \\ \displaystyle\sum_{k=1}^{N} w_k = 1. \end{cases} \tag{7.8}$$

The weights are constrained to sum to 1 instead of m, as the value of m does not affect the objective.

The optimal weights $w_k^\star$ are used to produce the final design. For a design with m points, the design is chosen to be the $\mathbf{z}_k$ (with corresponding $\mathbf{x}_k$) associated with the m largest values of $w_k^\star$. Alternatively, we can choose the design associated with the largest $w_k^\star$'s, allowing as many points as required to reach some weight goal, for example a total weight of 0.99. To notate the chosen design we simply re- index the points as $\{\mathbf{z}_j\}_{j=1}^m$.

Several common scalarizations of the design objective have been proposed that are more tractable [14]. The most commonly used is known as the *D-optimal design* and it minimizes the log-determinate.

$$\min_{\boldsymbol{\omega}} \ \log\det\left(\sum_{k=1}^{N} w_k \boldsymbol{\phi}(\mathbf{z}_k)\boldsymbol{\phi}^{\mathsf{T}}(\mathbf{z}_k)\right)^{-1} \\ \text{s.t.} \ \begin{cases} w_k \geq 0, \ k = 1, \ldots, N, \\ \displaystyle\sum_{k=1}^{N} w_k = 1. \end{cases} \tag{7.9}$$

This is a convex optimization problem. For any confidence percentage ρ, there is an ellipsoid centered at the least squares estimate of the model coefficients corresponding to a design, such that the true model coefficients are in the ellipsoid with a probability ρ. The D-optimal design corresponds to the minimum volume ellipsoid representing the confidence region for a fixed confidence level [30].

Another scalarization of the design problem is known as the *E-optimal design* where the matrix 2-norm (largest singular value) is minimized.

$$\min_{\boldsymbol{\omega}} \ \bar{\sigma}\left[\left(\sum_{k=1}^{N} w_k \boldsymbol{\phi}(\mathbf{z}_k)\boldsymbol{\phi}^{\mathsf{T}}(\mathbf{z}_k)\right)^{-1}\right] \\ \text{s.t.} \ \begin{cases} w_k \geq 0, \ k = 1, \ldots, N, \\ \displaystyle\sum_{k=1}^{N} w_k = 1. \end{cases} \tag{7.10}$$

This is a convex optimization problem, and furthermore can be recast as a semidefinite program (SDP).

$$\begin{aligned} &\max_{\boldsymbol{\omega},t} t \\ &\text{s.t.} \begin{cases} \displaystyle\sum_{k=1}^{N} w_k \boldsymbol{\phi}(\mathbf{z}_k)\boldsymbol{\phi}^{\mathsf{T}}(\mathbf{z}_k) \succeq t\mathbf{I} \\ w_k \geq 0,\ k = 1, \ldots, N, \\ \displaystyle\sum_{k=1}^{N} w_k = 1. \end{cases} \end{aligned} \tag{7.11}$$

The E-optimal design corresponds the confidence ellipsoid with its largest semi-axis minimized [30].

The last scalarization is known as *A-optimal design*, wherein the trace of the error covariance matrix is minimized.

$$\begin{aligned} &\min_{\boldsymbol{\omega}} \operatorname{Tr}\left[\left(\sum_{k=1}^{N} w_k \boldsymbol{\phi}(\mathbf{z}_k)\boldsymbol{\phi}^{\mathsf{T}}(\mathbf{z}_k)\right)^{-1}\right] \\ &\text{s.t.} \begin{cases} w_k \geq 0,\ k = 1, \ldots, N, \\ \displaystyle\sum_{k=1}^{N} w_k = 1. \end{cases} \end{aligned} \tag{7.12}$$

This is also a convex optimization problem, and can also be recast as an SDP.

$$\begin{aligned} &\max_{\boldsymbol{\omega},\mathbf{u}} \sum_{\ell=1}^{L} u_\ell \\ &\text{s.t.} \begin{cases} \begin{bmatrix} \displaystyle\sum_{k=1}^{N} w_k \boldsymbol{\phi}(\mathbf{z}_k)\boldsymbol{\phi}^{\mathsf{T}}(\mathbf{z}_k) & \mathbf{e}_\ell \\ \mathbf{e}_\ell^{\mathsf{T}} & u_\ell \end{bmatrix} \succeq 0,\ \ell = 1, \ldots, L \\ w_k \geq 0,\ k = 1, \ldots, N, \\ \displaystyle\sum_{k=1}^{N} w_k = 1. \end{cases} \end{aligned} \tag{7.13}$$

Here $\mathbf{e}_\ell$ is the ℓ^{th} standard basis vector in L dimensions, where L is the size of the surrogate fit basis vectors $\boldsymbol{\phi}(\mathbf{z})$. The A-optimal design minimizes the expected value of the Euclidean norm of error in the least squares estimate of the model coefficients [14].

It has been our experience that the results of all three of these scalarizations can be skewed if the sample is not centered. This can be easily corrected by translating the sample before converting to the surrogate basis. Let $\bar{\mathbf{z}} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{z}_k$ be the mean vector of the sample. Then simply replace each vector $\mathbf{z}_k$ with $\mathbf{z} - \bar{\mathbf{z}}$ in each of the optimizations.

As a toy example, take the polytope shown in Figure 7.7. A 1,000-point sample of the polytope is produced (using the gas dynamics point generation algorithm in Section 7.2.1). The resulting 15 point design selection is shown for the D-, E-, and A-optimal design optimizations. There are 6 points that show up in all three designs. The E- and A-optimal designs are the most similar, sharing 11 points.

The D-optimal design in Figure 7.7 visually seems to be the most spread out, with none of the points bunched up. However, the D-optimal took over 764 seconds to select, which was the longest optimization time of the three designs. The A-optimal design took 12.7 seconds to select and the E-optimal took 9.9 seconds. To further compare the quality of the designs, Section 7.5 will demonstrate some fitting examples using different designs and compare the fitting errors.

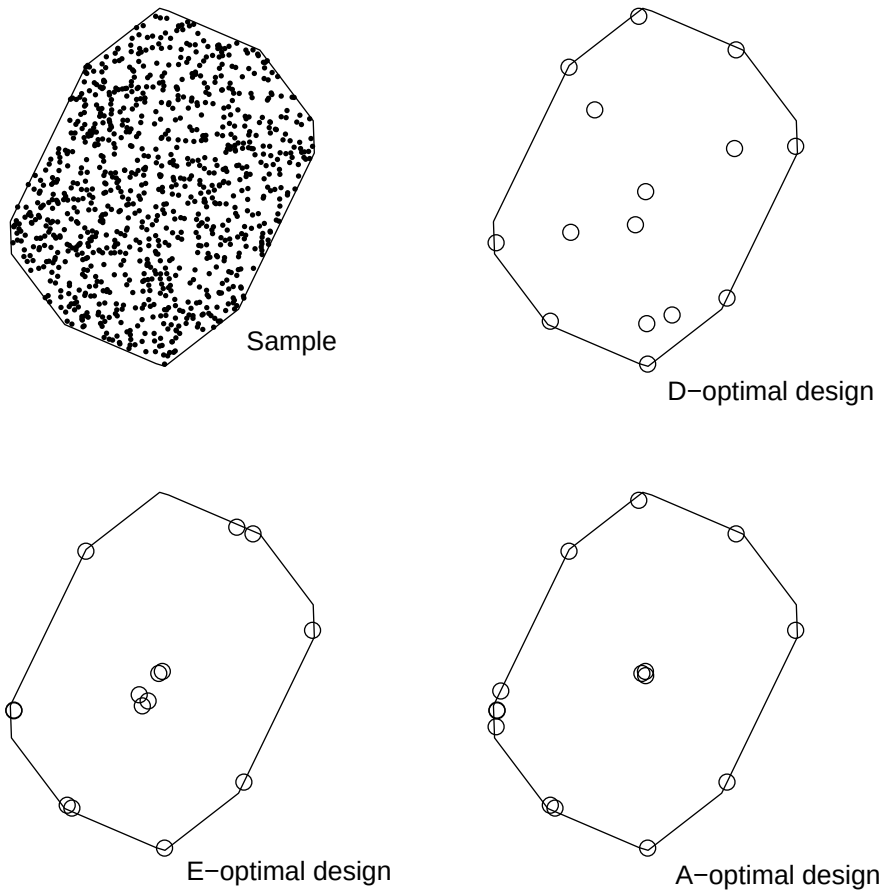

Figure 7.7: Selection of 15 design points using the D-, E-, and A-optimal design optimizations from a 1000 point sample of a 2-dimensional polytope.

7.4 Surrogate Fitting and Validation

The optimizations in Section 7.3 selects an experiment design $\{\mathbf{z}_k\}_{k=1}^m$ in a r-dimensional polytope from a large sample. The techniques for generating this sample (Section 7.2.1) provide corresponding points in the n-dimensional hyperrectangle $\mathcal{H}$, such that for every design $\mathbf{z}_k$ there is an $\mathbf{x}_k \in \mathcal{H}$ such that $\mathbf{z}_k = \mathbf{S}^\mathsf{T}\mathbf{x}$, where $\mathcal{R}(\mathbf{S})$ is the active subspace. Evaluation of the function of interest, f at the $\mathbf{x}_k$ vectors of parameters, yields the output $y_k = f(\mathbf{x}_k)$.

There is now enough information to create a surrogate fit of f. Recall, that $\boldsymbol{\phi}$ is the surrogate basis function — the function that maps a point in r- dimensions to the vector in the surrogate basis. Then the surrogate fit is the function with coefficients $\mathbf{a}$ that minimize the error on the design.

$$\min_{\mathbf{a}} \left\| \begin{bmatrix} \boldsymbol{\phi}^\mathsf{T}(\mathbf{z}_1) \\ \vdots \\ \boldsymbol{\phi}^\mathsf{T}(\mathbf{z}_m) \end{bmatrix} \mathbf{a} - \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \right\|. \tag{7.14}$$

The choice of norm does affect the fit. The 2-norm chooses coefficients to reduce the average error on the design points. This is known as the least-squares problem and is very efficient to solve for, and hence a quick calculation. The infinity-norm chooses coefficients to reduce the maximum error on the design points. This problem is solved with a linear program. Linear programs are also considered easy to solve, although the solution takes longer to compute than the least-squares solution. All examples throughout the rest of the document will use the 2-norm objective.

The optimization provides a surrogate fit on the r-dimensional subspace. To convert the fit to one in the full n dimensions, simply compose the fit in r dimensions with the linear transformation,

$$f_{\text{fit}}(\mathbf{x}) = \mathbf{a}^{\star\mathsf{T}}\boldsymbol{\phi}(\mathbf{S}^\mathsf{T}\mathbf{x}). \tag{7.15}$$

Validation of f_{fit} is then performed in the full n dimensions, and not on the subspace, as this is where the function will be used.

7.5 Toy Examples

In this section, we explore how various experiment designs affect surrogate fitting errors. Chapter 8 contains examples which demonstrate both the active subspace discovery and experiment design and their effect on the surrogate fit. This section will not explore how the choice of active subspace dimension affects the fit, but will focus on how changes in the design affect the fitting errors.

Let $\mathbf{S}$ be a 30×3 matrix with orthonormal columns. The columns of $\mathbf{S}$ span the active subspace of all of the following examples. We assume that this matrix is known.

The first example function f_1 is a quadratic function that depends only on the active subspace, and includes some noise.

$$f_1(\mathbf{x}) = \begin{bmatrix} 1 \\ \mathbf{S}^\mathsf{T}\mathbf{x} \end{bmatrix}^\mathsf{T} \mathbf{Q} \begin{bmatrix} 1 \\ \mathbf{S}^\mathsf{T}\mathbf{x} \end{bmatrix}^\mathsf{T} + \eta \tag{7.16}$$

where $\mathbf{Q}$ is a 4×4 symmetric matrix and η is Gaussian noise with zero mean and variance of 0.01.

Two 3000-point samples were created using the two methods described in Section 7.2. From these samples D-, E-, and A-optimal designs were selected using quadratic basis functions and the optimizations presented in Section 7.3. The D-optimal designs took an extraordinarily long time to compute. Table 7.1 shows the computation times.

Table 7.1: Running times to compute optimal designs from 3000-point samples for quadratic basis. The two sampling methods from Section 7.2 are shown. Designs were computed with MATLAB on a laptop running Windows Vista x64 with a 2GHz Intel Core 2 Duo processor.

	Sampling Method	
	1	2
D-optimal	666.3 min	787.9 min
E-optimal	7.5 min	8.1 min
A-optimal	9.2 min	9.2 min

Using these designs, a quadratic surrogate model that depends on the active subspace is fitted to the model. The fits were created to minimize the least-squares objective on the residual error. Error was assessed in the original 30-dimensional space using 600,000 validation points. Table 7.2 contains the average relative error over the validation points. As a comparison, an fitting to a random design produced a fitting error of 0.71%. The D-optimal design does not provide much better fitting errors than the E- or A-optimal and yet it took almost 100 times longer to compute. In this example, there isn't a significant difference between the two sampling methods.

Table 7.2: Average relative fitting errors for surrogates of f_1 computed using 6 optimal designs. Errors are assessed in the full 30-dimensional space over 600,000 validation points.

	Sampling Method	
	1	2
D-optimal	0.43%	0.45%
E-optimal	0.44%	0.45%
A-optimal	0.53%	0.48%

The second example is a function which is not purely quadratic.

$$f_2(\mathbf{x}) = \begin{bmatrix} 1 \\ \mathbf{S}^\mathsf{T}\mathbf{x} \end{bmatrix}^\mathsf{T} \mathbf{Q} \begin{bmatrix} 1 \\ \mathbf{S}^\mathsf{T}\mathbf{x} \end{bmatrix}^\mathsf{T} + 0.1 \left| \mathbf{x}^\mathsf{T}\mathbf{S}\mathbf{C} \cdot \operatorname{diag}(\mathbf{x}^\mathsf{T}\mathbf{S}\mathbf{S}^\mathsf{T}\mathbf{x}) \right| + \eta. \tag{7.17}$$

Here, $\mathbf{C}$ is an $r \times r$ matrix representing coefficients associated with cubic terms. This form was chosen for notational and computational simplicity. It contains cubic terms such as $\mathbf{x}_i^3$ and $\mathbf{x}_i^2\mathbf{x}_j$ but does not include any of the cubic terms of the form $\mathbf{x}_i\mathbf{x}_j\mathbf{x}_k$ for distinct i, j, and k. The same active subspace as the previous example is used, and therefore we can use the same designs. Table 7.3 contains the relative fitting errors for this function.

In this example, the D-optimal designs did worse than the E-, or A-optimal designs. However, it should be noted that a random design produced a fitting error of 1.2%. This may be due to the fact that f_2 is not quadratic, and the design was specifically created to fit quadratics well. In this sense, the experiment design technique only works if the function in question if very close to quadratic. Section 7.6.2 will discuss some methods for expanding a design to include more points.

Table 7.3: Average relative fitting errors for surrogates of f_2 computed using 6 optimal designs. Errors are assessed in the full 30-dimensional space over 600,000 validation points.

	Sampling Method	
	1	2
D-optimal	2.0%	3.4%
E-optimal	1.1%	2.0%
A-optimal	1.2%	2.3%

The last example is a quadratic function with noise that does not depend solely on its active subspace.

$$f_3(\mathbf{x}) = \left[\begin{smallmatrix}1\\ \mathbf{S}^\mathsf{T}\mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Q}_1 \left[\begin{smallmatrix}1\\ \mathbf{S}^\mathsf{T}\mathbf{x}\end{smallmatrix}\right]^\mathsf{T} + 0.01\mathbf{x}^\mathsf{T}\mathbf{S}_\perp\mathbf{Q}_2\mathbf{S}_\perp^\mathsf{T}\mathbf{x} + \eta. \tag{7.18}$$

Here $\mathbf{Q}_1$ is an $r+1 \times r+1$ matrix as before, and $\mathbf{Q}_2$ is an $n-r \times n-r$ matrix. The function f_3 most strongly depends on the subspace $\mathcal{R}(\mathbf{S})$ but it also depends on the orthogonal subspace $\mathcal{R}(\mathbf{S}_\perp)$. Table 7.4 contains the relative fitting errors associated with the various designs. For comparison, a fit made with a random design had a relative error of 4.8%.

Table 7.4: Average relative fitting errors for surrogates of f_3 computed using 6 optimal designs. Errors are assessed in the full 30-dimensional space over 600,000 validation points.

	Sampling Method	
	1	2
D-optimal	3.6%	3.8%
E-optimal	3.6%	4.0%
A-optimal	3.6%	3.9%

Based on these three simple examples, it appears that the large amount of time spent on computing the D-optimal design is not worth the results it produces. Sampling method 2 produced fitting errors that were no better, and often worse than

that of sampling method 1. As seen in Section 7.2.3, this may be do to the fact that method 2 might not produce samples that are outside of the polytope $\mathcal{P}_1$ and therefore no improvement is made. The next section will discuss some improvements that could be made upon these methods.

7.6 Method Improvements

7.6.1 Sampling on the Polytope Boundaries

The optimization problems in Section 7.3 that select the experiment design can be very large and take a long time to solve. Each formulation involves the weighted sum of N $L \times L$ matrices, where N is the number of sample points, and L is the number of basis functions for a fit on the lower dimension r. For example, say the sample involves 30,000 points, and the fit is a quadratic in 5 variables. Then each optimization involves the weighted sum of 30,000 21×21 matrices. These large dimensions affect computation times considerably; however, large sample sizes are needed to adequately fill the polytope.

It has been our observation that most of the optimizations involving a quadratic basis result in a design of points on the boundaries of the polytope and a point in the center of the polytope. It therefore makes sense to only sample the boundaries of the polytope and the center point. In this case, the sample would not need to be as large in order to adequately find good design points. The following are two methods for sampling the boundaries, analogous to the two methods of sampling presented in Section 7.2.

To gain the center point of the polytope, transform the center point $\mathbf{x}_0$ of $\mathcal{H}$ to the $\mathbf{z}$-space, via, $\mathbf{z}_0 = \mathbf{S}^\mathsf{T}\mathbf{x}_0$. Choose a vector $\mathbf{v} \in \mathbb{R}^r$ from a distribution that is uniform in direction. This is achieved by sampling a Gaussian distribution and normalizing [66, 73]. A sample point on the boundary $\mathbf{z}_k$ is achieved by finding the maximum γ such that $\mathbf{z}_0 + \gamma\mathbf{v}$ is in the polytope. This procedure is then repeated, picking a new direction each time, to produce a sample on the polytope boundary.

When discussing the two original sampling methods, Section 7.2.3 noted that each sampled a different polytope. Method 1 sampled the intersection of the subspace and $\mathcal{H}$, which was denoted $\mathcal{P}_1$. Method 2 sampled the projection of $\mathcal{H}$ onto the subspace, a polytope denoted $\mathcal{P}_2$. Sampling the boundaries of these polytopes requires one of two optimizations. After the random direction $\mathbf{v}$ is sampled, a point on the boundary of $\mathcal{P}_1$ corresponds to

$$\begin{aligned} &\max_{\gamma} \ \gamma \\ &\text{s.t. } \mathbf{x}_0 + \gamma\mathbf{S}\mathbf{v} \in \mathcal{H}. \end{aligned} \tag{7.19}$$

The boundary point is $\mathbf{z}_0 + \gamma^\star \mathbf{v}$. This is a very simple optimization, and only requires some simple vector operations to solve. To find a boundary point of $\mathcal{P}_2$ requires a linear program.

$$\begin{aligned} &\max_{\mathbf{x},\gamma} \ \gamma \\ &\text{s.t.} \ \begin{cases} \mathbf{x} \in \mathcal{H}, \\ \mathbf{z}_0 + \gamma \mathbf{v} = \mathbf{S}^\mathsf{T} \mathbf{x}. \end{cases} \end{aligned} \tag{7.20}$$

This optimization solves for the furthest possible distance traveled along the direction $\mathbf{v}$ such that there is a corresponding $\mathbf{x} \in \mathcal{H}$. The resulting boundary point is $\mathbf{S}^\mathsf{T}\mathbf{x}^\star$.

As discussed previously, $\mathcal{P}_2$ is larger than $\mathcal{P}_1$ and so it is beneficial to sample its boundary. However, the boundary points do take slightly more time to compute. A simple example in two dimensions in shown in Figure 7.8 along with the resulting D-optimal designs. In the case of designing on $\mathcal{P}_1$, a much smaller sample than was demonstrated in Section 7.2.1 yields the same end design. However, the optimization takes much less time, due to the decrease in sample size.

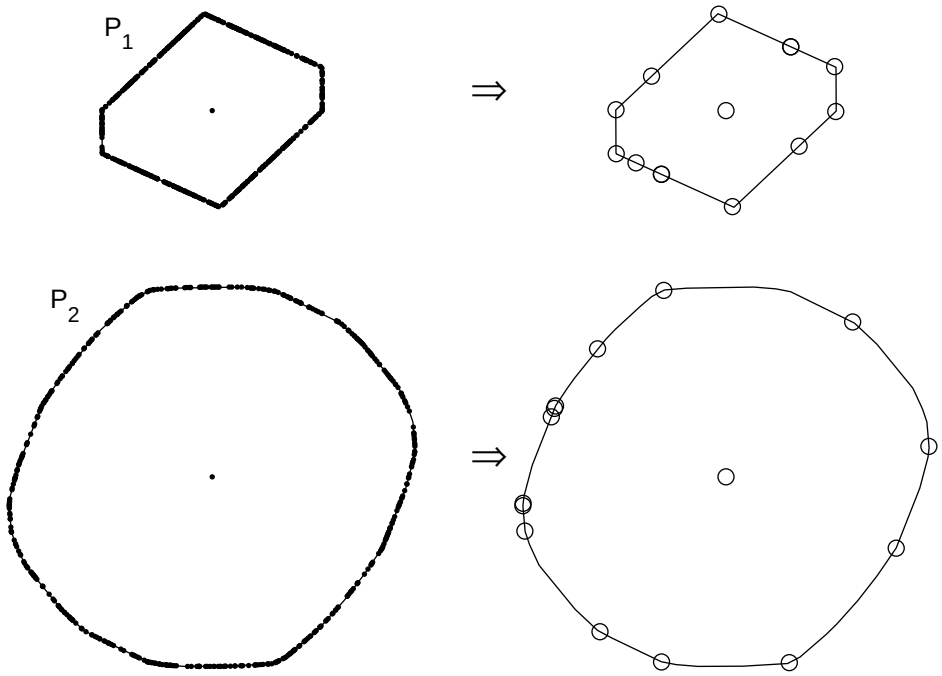

Figure 7.8: 500 point samples of the boundaries of $\mathcal{P}_1$ and $\mathcal{P}_2$ (and the center points) and the resulting D-optimal designs.

As an example, take the function f_1 from equation (7.16). The fitting errors for a quadratic surrogate created from designs on the boundaries of $\mathcal{P}_1$ and $\mathcal{P}_2$ are about the same as they were before. However, the design optimization time was significantly decreased. Since the initial samples are all on the polytope boundaries, only 500 are chosen. Table 7.5 shows the optimization times.

Table 7.5: Running times to compute optimal designs from 500-point samples for quadratic basis. The two samples used are on the boundaries of the polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$. Designs were computed with MATLAB on a laptop running Windows Vista x64 with a 2GHz Intel Core 2 Duo processor.

	Sampling Method	
	1	2
D-optimal	306.9 sec	212.4 sec
E-optimal	1.6 sec	1.7 sec
A-optimal	7.5 sec	10.1 sec

7.6.2 Expanding a Design

The design optimization problems provide a weight for each point in the polytope sample. After about 99% of the total weight has been distributed to the primary design points, the design starts to bunch several points together. This may make sense in laboratory experiments where repeating an experiment at the same parameter values may yield different results. However, for the computer experiments used for surrogate fitting, repeated points will only bias the fit to be better at those locations in the parameter space. However, it may be necessary to increase the number of design points if, for example, f is not similar to the form of the surrogate fit. Many samples may help catch larger trends in the function.

One idea would be to perform designs on both types of Polytopes. Since $\mathcal{P}_1 \subseteq \mathcal{P}_2$, designs on the boundaries of $\mathcal{P}_1$ and $\mathcal{P}_2$ will be disjoint (except for the center point). If the boundaries of $\mathcal{P}_1$ and $\mathcal{P}_2$ are far enough apart, a union of the two designs will provide parameter vectors that are spread out. Figure 7.9 shows an example of this type of design union.

Another possible way to expand a design would be to use Voronoi decomposition. Given an already optimized design, the Voronoi decomposition divides the space into polyhedra, such that there is one design point in each polyhedron, and all points in the interior of any polyhedron are closer to the enclosed design point than any other design point. The vertices of the polyhedron provide a good addition to the design as they are equidistant from the $r + 1$ closest design points. In this sense, the Voronoi vertices are not dense with respect to the current design, although they can be close to each other. The biggest problem with this approach is that the number of Voronoi vertices grows very rapidly[3] and can take some time to compute and may provide more points than desired. Figure 7.10 shows the Voronoi cells for a design in 2- dimensions.

[3]Number of Voronoi vertices with m design points in r dimensions is in $O(m^{\lceil r/2 \rceil}/\lceil r/2 \rceil!)$ [55].

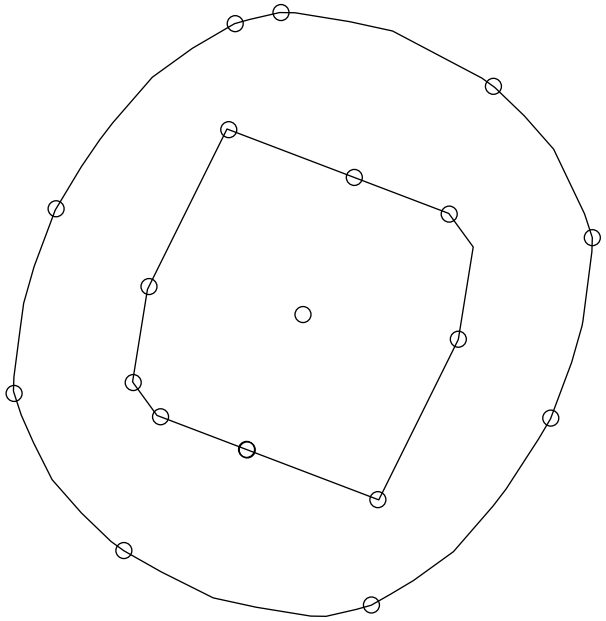

Figure 7.9: The union of designs on the boundaries of polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$ along with the center point for a toy problem.

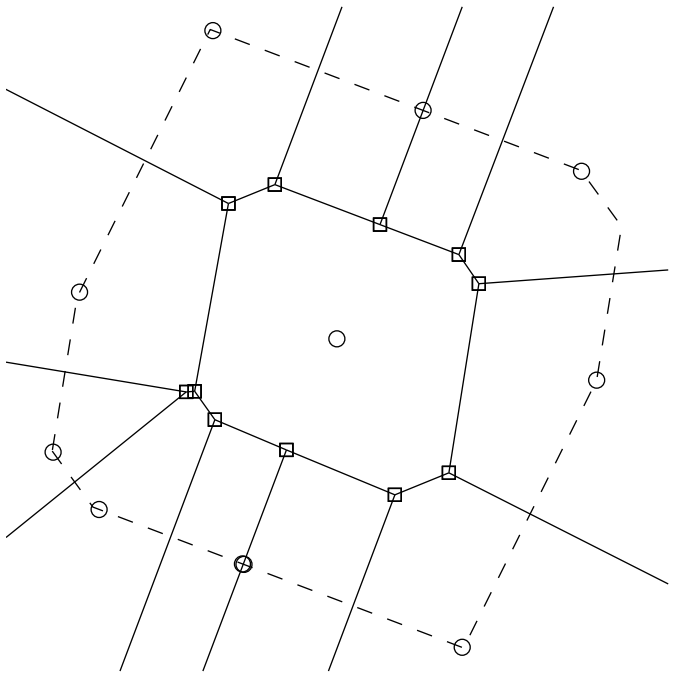

Figure 7.10: Voronoi cells corresponding to a design on a polytope. Polytope is shown as dashed lines. Circles designate the original points and squares the Voronoi vertices.

7.6.3 Including Orthogonal-Space Dependence

The technique developed in this chapter creates surrogate models that depend solely on an active subspace $\mathcal{R}(\mathbf{S})$. The assumption is that the number of basis functions grows unfavorably with the dimension, and hence fitting only on the active subspace decreases the number of evaluations required substantially. However, the function f does not necessarily depend solely on the active subspace $\mathcal{R}(\mathbf{S})$. The term "active" simply implies that the majority of the dependence occurs along the subspace directions. However, there may be some dependence in the orthogonal subspace $\mathcal{R}(\mathbf{S}_\perp) = \mathcal{N}(\mathbf{S}^\mathsf{T})$.

To capture some of that dependence without too many more function evaluations we allow the surrogate fit to include linear terms in the orthogonal directions. If $\mathbf{S}_\perp$ is the $n \times n - r$ matrix whose orthonormal columns span the space orthogonal to the active subspace, then the surrogate model formulation will be

$$f_{\text{fit}}(\mathbf{x}) = \mathbf{a}^\mathsf{T}\boldsymbol{\phi}(\mathbf{S}^\mathsf{T}\mathbf{x}) + \mathbf{b}^\mathsf{T}\mathbf{S}_\perp^\mathsf{T}\mathbf{x} \tag{7.21}$$

where $\mathbf{a}$ and $\mathbf{b}$ are the surrogate coefficients. There are only $n - r$ more coefficients in this formulation than that of equation (7.15), and therefore it only requires on the order of n more function evaluations.

Creating an experiment design for this fit is done in two steps. First create a design as before without any orthogonal component considered. Second, simply add points to the design that have nonzero orthogonal subspace components. For example, these added points could come from an $n - r$ point, n-dimensional Latin-hypercube design. For another example, take the hyperrectangle vertices corresponding to the orthants containing the columns of $\mathbf{S}_\perp$. We do not suggest optimizing this portion of the design because presumably the dimension $n - r$ is large enough that it is very difficult to effectively sample the space.

This concept of allowing the surrogate fit to depend on the space orthogonal to the active subspace, can be extended to allow any basis whose size grows linearly with dimension. For example, we can allow both linear and purely quadratic terms (no cross terms) in the orthogonal directions. This equates to creating a fit of the form

$$f_{\text{fit}}(\mathbf{x}) = \mathbf{a}^\mathsf{T}\boldsymbol{\phi}(\mathbf{S}^\mathsf{T}\mathbf{x}) + \mathbf{b}^\mathsf{T}\mathbf{S}_\perp^\mathsf{T}\mathbf{x} + \mathbf{x}^\mathsf{T}\mathbf{S}_\perp\boldsymbol{\Lambda}\mathbf{S}_\perp^\mathsf{T}\mathbf{x} \tag{7.22}$$

where $\boldsymbol{\Lambda}$ is a diagonal $n - r \times n - r$ matrix.

Chapter 8 provides an example of this technique.

Chapter 8

Examples Using Active Subspace Methods

This chapter presents some examples combining the active subspace discovery algorithm of Chapter 6 and the subsequent design and fitting techniques of Chapter 7. The first couple of examples will explore the effect of choosing different subspace dimensions and different designs on fitting quadratic surrogate models. The last example will using automated active subspace discovery as part of a dataset consistency measure calculation.

8.1 Methane Combustion

GRI-Mech 3.0 [106] target CH3.C1a is the maximum CH3 concentration in a shock tube oxidation of methane [18]. Simulation code for this observable has 313 parameters. The domain of our analysis will allow each variable to vary a factor of 2 above and below its nominal value.

Section 6.6.3 briefly examined this observable. Recall that a basic sensitivity analysis from linear fits of a Hadamard design showed an asymptotic decay of sorted parameter sensitivities. The GRI-Mech project uses the top 11 ranked parameters as the active parameters for this observable. For this example, the top 100 ranked parameters are chosen as the active parameters. The remaining 213 parameters are set at their nominal values. The rest of the analysis in this section will only allow these 100 parameters to vary; all subspace discovery analysis, surrogate fitting, and fit validation will be on the active parameters.

Let $M : \mathbb{R}^{100} \to \mathbb{R}$ be the function that takes the $\log_{10}$ of the 100 active parameters to the $\log_{10}$ of target CH3.C1a. Each evaluation of M takes about 0.26 seconds on a laptop running Windows Vista x64 with a 2GHz Intel Core 2 Duo processor. A quadratic surrogate in 100 parameters has 5,151 coefficients. The required 5,151 simulations would take about 22 minutes to run, which isn't too long. However, simulation of some of the GRI-Mech targets can take as long as half an hour per run [41]. Analysis of target CH3.C1a may provide some insight into how surrogate fitting with active subspace techniques will work with other GRI-Mech targets.

The first step in creating a surrogate model using active subspace techniques is using the active subspace discovery algorithm (Algorithm 6.1 on page 56). Figure 8.1 shows the singular values of the gradient matrix (computed using numerical gradients) after 40 iterations of the algorithm. The singular values of the gradient matrix imply that M depends on a lower dimensional active subspace. The dimension of the subspace could be taken as low as 1 or as high as 30 or 35.

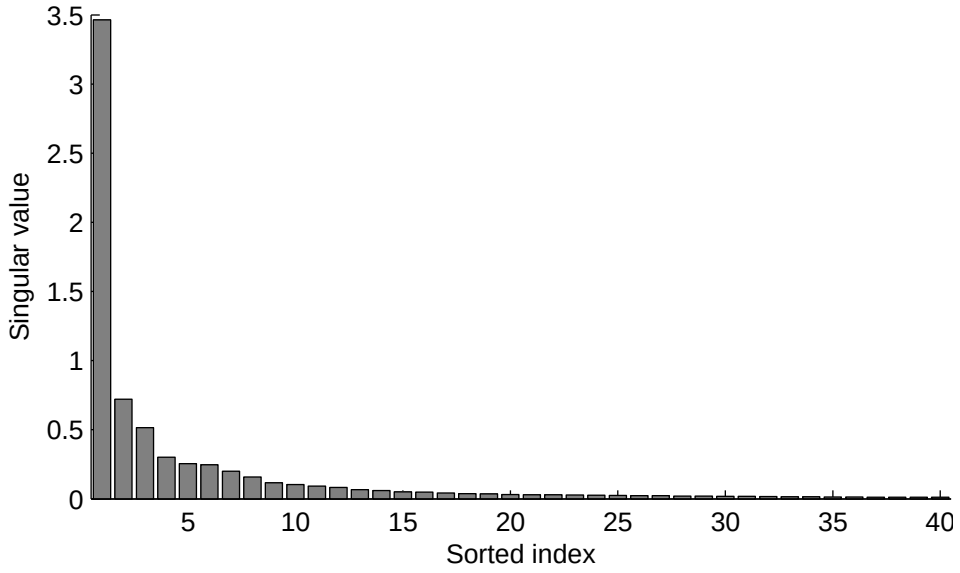

Figure 8.1: Singular values of gradient matrix after 40 iterations of the active subspace discovery algorithm for GRI-Mech 3.0 target CH3.C1a

To examine the effects of using an active subspace, quadratic surrogate models are created using several active subspace dimensions. Fitting designs were created using an E-optimal choice from a sample created using sampling Method 2 (see §7.2.1 and §7.3). The size of each design was double the number of coefficients of a quadratic in the subspace dimension. Table 8.1 shows the relative fitting error of M for several subspace dimension choices as well as the number of function evaluations needed to generate the quadratic. The errors are relative and assessed in the log coordinates on a 40,000-point validation sample in all 100 active parameters. The total evaluations and CPU time shown reflect both the evaluations used for fitting as well as the amount of evaluations used for the active subspace discovery, given that the algorithm stopped after one more iteration beyond that of the dimension of the active subspace (see §6.3).

The total number of evaluations also takes into account the fact that several points are reused from the subspace discovery algorithm for fitting. For example, discovering a 6-dimensional subspace required 7 groups of 101 evaluations, and fitting a quadratic in 6 dimensions used 49 evaluations (twice the number of coefficients minus 7 points reused from the subspace discovery) for a total of 756.

Table 8.1: Surrogate fitting errors for GRI-Mech 3.0 target CH3.C1a. Several different active subspace dimension are shown, as well as a fit in all 100 active variables. Fitting errors are relative and assessed on the same 40,000-point validation sample of the full 100 dimensions.

Fit Descr.	Avg. Err.	Max. Err.	Total Evals.	CPU Time (min)
Full Dimension	0.07%	0.56%	10,302	44.6
1D Subspace	0.37%	2.9%	206	0.89
2D Subspace	0.66%	6.0%	312	1.35
4D Subspace	0.38%	3.4%	530	2.30
6D Subspace	0.19%	1.7%	756	3.28
10D Subspace	0.20%	1.4%	1,232	5.34
20D Subspace	0.09%	0.74%	2,562	11.1
30D Subspace	0.08%	0.70%	4,092	17.7

From the table we can see that there is not a significant decline in fitting error from a fit in all 100 dimensions when using a 6- or 10-dimensional subspace. However, there is an order of magnitude decrease in the number of function evaluations required, and hence the computation time. Had the target simulation taken much longer than it did, fitting a surrogate model in 100 variables would be very time consuming.

As noted in Section 7.6.3, it does not require many more function evaluations to add linear terms in the direction orthogonal to the active subspace. To create these fits a 200-point Latin hypercube design of the 100-dimensional domain was added to the experiment design to include points not on the active subspace. Table 8.2 shows the surrogate fitting errors associated with these types of fits. These errors were assessed on the same 40,000-point validation as before. Furthermore we can add terms for the "pure" quadratic monomials in the directions orthogonal to the active subspace, i.e. those of the form x_i^2 and not those of the form $x_i x_j$. Again, the number of these monomials is linear in the number of dimensions. A 400-point Latin hypercube design was added to the experiment design (200 for the linear terms and 200 for the pure quadratic terms). Table 8.3 shows the surrogate fitting errors when including both the linear and purely quadratic terms in the orthogonal directions. In both cases, adding the extra terms for the orthogonal subspace does not lead to much improvement. This is most likely due to the fact that the active subspaces were correctly computed, and full quadratic is needed in those directions.

Table 8.2: Surrogate fitting errors for GRI-Mech 3.0 target CH3.C1a. Several different active subspace dimension are shown, as well as a fit in all 100 active variables. Each fit (other than the fit in the full 100 dimensions) is quadratic on the active subspace, plus includes linear terms for directions orthogonal to the active subspace. Fitting errors are relative and assessed on the same 40,000-point validation sample of the full 100 dimensions.

Fit Descr.	Avg. Err.	Max. Err.	Total Evals.	CPU Time (min)
Full Dimension	0.07%	0.56%	10,302	44.6
1D Subspace	0.35%	2.2%	404	1.75
2D Subspace	0.32%	2.3%	508	2.20
4D Subspace	0.32%	2.1%	722	3.13
6D Subspace	0.20%	1.3%	944	4.09
10D Subspace	0.18%	1.4%	1,412	6.12
20D Subspace	0.09%	0.72%	2,722	11.8
30D Subspace	0.07%	0.57%	4,232	18.3

Table 8.3: Surrogate fitting errors for GRI-Mech 3.0 target CH3.C1a. Several different active subspace dimension are shown, as well as a fit in all 100 active variables. Each fit (other than the fit in the full 100 dimensions) is quadratic on the active subspace, plus includes linear and purely quadratic terms for directions orthogonal to the active subspace. Fitting errors are relative and assessed on the same 40,000-point validation sample of the full 100 dimensions.

Fit Descr.	Avg. Err.	Max. Err.	Total Evals.	CPU Time (min)
Full Dimension	0.07%	0.56%	10,302	44.6
1D Subspace	0.35%	2.3%	602	2.61
2D Subspace	0.32%	2.3%	704	3.05
4D Subspace	0.29%	2.5%	914	3.96
6D Subspace	0.18%	1.0%	1,132	4.91
10D Subspace	0.17%	1.1%	1,592	6.90
20D Subspace	0.09%	0.73%	2,882	12.5
30D Subspace	0.08%	0.72%	4,372	18.9

8.2 Cellular Calcium Response

The next example returns to the biological signal application introduced in Section 6.6.4. The model simulates the calcium response to application of the ligand C5a in a murine macrophage cell line [62]. Recall that the model involves 8 coupled ODEs with 34 uncertain parameters and has several features that were previously examined. For this example, we examine the maximum calcium concentration response to a 100nM ligand application.

Section 6.6.4 showed the results from the subspace discovery algorithm for this model. The active subspace is somewhere between 1 and 3 dimensions depending on choice of threshold. A quick look at the surface of this function along the 1-dimensional active subspace shows that the model is not well approximated by a quadratic function. However, if the model is well approximated by a piecewise quadratic, then the Data Collaboration branch and bound techniques can be utilized for analysis. Therefore, in this section the focus is to create a quadratic surrogate model over a smaller domain. Nominal values for each parameter are given in the original work [62] and a 20% uncertainty for each parameter is suggested in [34]. For the example in this section we focus on a domain centered at the nominal values and allowing just 2% variation in each parameter.

A quadratic surrogate function was fit on this smaller domain. The fit maps the $\log_{10}$ of the 34 parameters to maximum calcium concentration (without a logarithmic transformation). Table 8.4 shows the errors for several surrogate fits, including a few different active subspace dimensions. Errors for fits using linear and pure quadratic terms in the directions orthogonal to the active subspace are also shown. The errors are assessed on a 13,600-point validation set. The errors are relative to the range of the observable — the difference between the maximum and minimum values of the observable.

Table 8.4: Fitting errors for the Lemon *et. al.* [62] model for maximum calcium concentration due to a 100nM ligand application. Errors are relative to the range of the function over the parameter domain.

	$\mathbf{S}_\perp$	Error		Total	CPU Time
Fit Descr.	Terms	Avg.	Max.	Evals.	(sec)
Full Dimension	—	0.58%	10.7%	1,260	327
1D Subspace	None	0.51%	16.3%	74	19.2
2D Subspace	None	0.31%	15.5%	114	29.6
3D Subspace	None	0.30%	15.0%	156	40.6
1D Subspace	Lin.	0.52%	11.9%	140	36.4
2D Subspace	Lin.	0.36%	12.6%	178	46.3
3D Subspace	Lin.	0.50%	8.58%	218	56.7
1D Subspace	Lin.+Pure Quad	0.42%	10.1%	206	53.6
2D Subspace	Lin.+Pure Quad	0.51%	8.55%	242	62.9
3D Subspace	Lin.+Pure Quad	0.38%	10.7%	280	72.8

8.3 Consistency of Cellular Calcium Response Dataset

In [34], Ryan Feeley demonstrated the consistency of a dataset made from several features of the calcium response model by Lemon *et. al.* [62] used in the previous section. The dataset included the rise time, peak offset, and fall time of the calcium response due to each of six different ligand doses. When computing the consistency measure, quadratic surrogate models were built for each of the 18 attribute models. This was done with an iterative fitting algorithm that continually adds points to the regression until each fit "settles" (see §B.5.2).

The goal of the active subspace method of surrogate fitting is to reduce the amount of evaluations required. To fairly compare results of a consistency measure computation when the active subspace method is used for surrogate fitting to the computation without the active subspace method, the iterative fitting algorithm is not used. Instead, each fit is created with a fixed number of evaluations that depends on the number of dimensions. This calcium response model is generally not well fit by quadratics, so we examine a smaller example than Feeley discusses. In this way, the methods can be compared with the difficulties associated with the model playing less of a role.

The dataset contains four models: the peak offset of the calcium response from each of the ligand doses 50, 100, 250, and 500nM. The models each have 34 uncertain parameters. The ranges of the parameters were taken at 2% above and below literature values. The experiment data was taken from the Alliance for Cellular Signaling (AfCS) [3, 111] as in [34]. However, it should be noted that this data comes from a cell type that is different than that for which the model was designed. Therefore, the results presented here should only be viewed as an example of the techniques and not as any conclusions about the model or data.

Five iterations of the branch and bound algorithm were used to compute the consistency measure of this dataset. Surrogate models were generated to reduce the infinity-norm of the residual error in the experiment designs.[1] The resulting bounds on the consistency measure are $[-0.0583,\ 1.40]$ which took a total of 30.2 hours to compute on a desktop computer running Linux. These bounds do not imply anything about the consistency of the dataset, since the measure can be either positive or negative. More branch and bound iterations would be required to resolve the consistency or inconsistency of the dataset.

The computation was repeated, this time using the active subspace method to generate the surrogate models. A gradient matrix singular value threshold of 0.05 was used (see Algorithm 6.1). For experiment designs, sampling method 1 was used to generate a sample on the subspace polytope from which the E-optimal design

[1] The infinity-norm was minimized since the maximum error is what is added to the experiment uncertainty for the consistency measure calculation.

was computed. The resulting bounds on the consistency measure were [0.0948, 2.16] which took a total of 15.3 hours to compute. In this case, even though the upper bound on the consistency was larger than before, the lower bound was positive, and hence the dataset is consistent. The active subspace for each model, in each iteration, was automatically computed to be 1-dimensional.

The difference in the results suggests the active subspace method works for this example. The computation using the method took about half as long as the computation without it, and produced a better lower bound. It did, however, produce a worse upper bound on the consistency measure. Table 8.5 shows the number of evaluations used for the consistency measure computation with and without the active subspace algorithm. Similarly, maximum fitting errors are shown. The computation with the active subspace algorithm used about half as many evaluations, which explains the difference in computation time. Note that each computation used the same number of evaluations for validation since fitting errors are assessed on the full 34 dimensions. If it were not for the large number of validation points, the time difference between the two methods would have been larger.

The fitting errors are better using the active subspace method for three of the four surfaces. In two cases (the 250nM and 500nM ligand doses) the errors are an order of magnitude better. The large error on the fit for the model with a 100nM ligand dose is likely the reason that the upper bound on the consistency measure is larger when the active subspace method is used. More iterations of the branch and bound algorithm would only improve the fitting errors, and hence the bounds on the consistency measure.

Table 8.5: Number of evaluations and fitting errors for the consistency of the calcium response dataset with and without the active subspace method. Totals are across all 5 iterations of the branch and bound algorithm. Fitting errors are absolute, and can be compared to a maximum value of about 0.11 for each model.

	w/o subspace	w/ subspace
# evals. for subspace discovery	0	1,400
# evals. for fit	252,000	40,860
# evals. for validation	126,000	126,000
Total # of evals	378,000	168,260
Max. fit error (50nM model)	0.151	0.103
Max. fit error (100nM model)	0.121	0.309
Max. fit error (250nM model)	0.121	0.0164
Max. fit error (500nM model)	0.108	0.0121

Chapter 9

Other Applications for the Data Collaboration Methodology

The Data Collaboration methodology, as described in Chapter 2, has been used primarily for checking dataset consistency [32, 33] and making model predictions [42, 43]. In Chapter 5, a method for finding a representative feasible point is described. This chapter will described several other tools and techniques for exploring a feasible set as well as methods for using the model prediction techniques in new ways.

9.1 Sampling the Feasible Set

The feasible set associated with the Data Collaboration setup is often made of nonlinear constraints in many variables. As such, traditional sampling techniques such as sampling from uniform or Gaussian distributions in n dimensions do not work as they do not take into account constraints that are not coordinate aligned. Even a technique such as rejection sampling can be frivolous in high dimensions (see §10.1). In this section, we modify the gas dynamics sampling algorithm for polytopes (Algorithm 7.1 in Section 7.2.1) to sample a set described by nonconvex quadratic constraints. The Data Collaboration techniques involve fitting model constraints with quadratic constraints thus creating such a set.

The major difference between the algorithm presented here and the one previously described for sampling a polytope have to do with the computation of gas particle reflections off of the set boundaries. Algorithm 9.1 shows the steps in the procedural form, including the computation required for reflection when the constraints are nonconvex quadratic.

Algorithm 9.1 Gas Dynamics Sampling Algorithm for Nonconvex Quadratically Constrained Sets

Require: $\mathbf{x}(0)$ {Starting feasible point}
Require: N {Number of desired samples}
Require: $\{\mathbf{Z}_i\}_{i=1}^m$ {Matrices defining constraints $\left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \leq 0$}

for $k = 1$ to N **do**
  Choose $\mathbf{v} \in \mathbb{R}^n$ from a Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ {"velocity"}
  $T_{\text{remain}} \leftarrow 1$ {Remaining "time" to next sample}
  $\mathbf{x}_{\text{now}} \leftarrow \mathbf{x}(k-1)$ {Current particle position}
  cnt $\leftarrow 0$ {Number of reflections for this point}
  while $T_{\text{remain}} > 0$ **do**
    $\{T^\star, i^\star\} \leftarrow \min_i \max_T T$ s.t. $\left[\begin{smallmatrix}1\\ \mathbf{x}_{\text{now}}+T\mathbf{v}\end{smallmatrix}\right]^{\mathsf{T}} \mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}_{\text{now}}+T\mathbf{v}\end{smallmatrix}\right] \leq 0.$
    if $T^\star > T_{\text{remain}}$ {Didn't hit boundary} **then**
      $\mathbf{x}(k) \leftarrow \mathbf{x}_{\text{now}} + T_{\text{remain}}\mathbf{v}$
      $T_{\text{remain}} \leftarrow 0$
    else if $T^\star \geq 0.99 T_{\text{remain}}$ {Near boundary} **then**
      $\mathbf{x}(k) \leftarrow \mathbf{x}_{\text{now}} + 0.99 T_{\text{remain}}\mathbf{v}$
      $T_{\text{remain}} \leftarrow 0$
    else if cnt$= 6$ **then**
      $\mathbf{x}(k) \leftarrow \mathbf{x}_{\text{now}} + 0.99 T^\star\mathbf{v}$
      $T_{\text{remain}} \leftarrow 0$
    else
      $\mathbf{x}_{\text{now}} \leftarrow \mathbf{x}_{\text{now}} + 0.999\alpha^\star\mathbf{v}$
      $T_{\text{remain}} \leftarrow T_{\text{remain}} - T^\star$
      $\boldsymbol{\eta} \leftarrow \left[\mathbf{0}_{n\times 1} \quad I_{n\times n}\right] \mathbf{Z}_{i^\star} \left[\begin{smallmatrix}1\\ \mathbf{x}_{\text{now}}\end{smallmatrix}\right]$ {Boundary normal vector}
      $\mathbf{v} \leftarrow \mathbf{v} - 2\frac{\boldsymbol{\eta}^{\mathsf{T}}\mathbf{v}}{\boldsymbol{\eta}^{\mathsf{T}}\boldsymbol{\eta}}\boldsymbol{\eta}$ {Reflect velocity vector}
    end if
    cnt $\leftarrow$cnt$+1$
  end while
end for
return $\{\mathbf{x}(k)\}_{k=0}^N$

Several things should be noted in Algorithm 9.1. If the sampled set is convex, the sample will converge to a uniform sampling of the set as the number of samples goes to infinity [115]. Without the convexity, the uniformity of the sample is not necessarily achieved. Furthermore, the variance of the velocity (line 2) and the time between recorded samples (line 3) can affect the time the algorithm takes to achieve a satisfactory spread of points. What dictates what variances and times are "too large" and "too small" depends on the set itself, and at this point we do not have a good way to determine a good distance for nonconvex sets. However, to combat the problem of too many reflections, Algorithm 9.1 limits the number of reflections to 6 as before (line 14).

This algorithm can be readily generalized to any set whose boundaries have easily computed gradients. The gradients are needed to compute reflection angles. As the Data Collaboration feasible set includes nonconvex quadratic constraints (with surrogate modeling), the algorithm, as shown, is sufficient.

9.2 A Posteriori Parameter Ranges

The prior information hyperrectangle ($H = \{\mathbf{x} : \alpha_i \leq x_i \leq \beta_i\}$) is the set of coordinate aligned bounds on the parameter values, and represents the knowledge of each parameter in an uncorrelated fashion. Once the parameters are further constrained by the experimental models, data, and uncertainty, it may be possible to shrink the bounds on some of the parameters and still maintain the same feasible set. These new ranges are known as the *a posteriori* parameter ranges (the term coming from philosophy [7] and Bayesian statistics [61]).

In the context of Data Collaboration the problem of finding posterior ranges is posed as the question: *What are the smallest coordinate aligned cube that contains the feasible set?* Or more specifically, for a single parameter: *What is smallest and largest values that a parameter can take in the feasible set?* The Data Collaboration methods can answer these questions via model predictions.

For each parameter x_i, define a prediction model $M_{0,i}(\mathbf{x}) = x_i$. Solve the model prediction problem (see §2.3) to obtain the lower bound $L_{0,i}$ and upper bound $R_{0,i}$. The interval $[L_{0,i},\ R_{0,i}]$ is the posterior range of x_i.

We performed this calculation for the GRI-Mech 3.0 dataset (described in Section 2.6). Figure 9.1 shows the percent reduction in parameter interval length that results from these calculations. About 20 of the parameters show an interval length reduction.

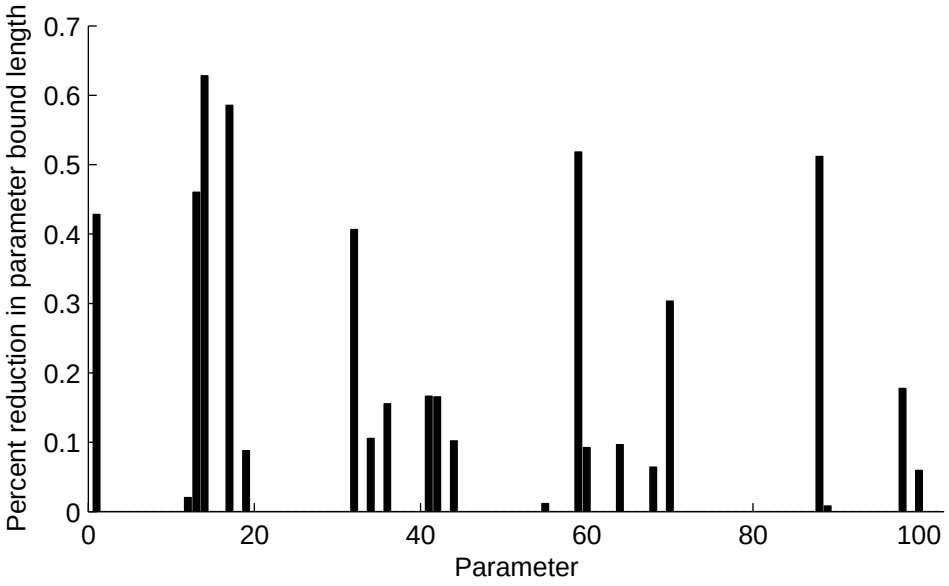

Figure 9.1: Percentage reduction in interval length when shrinking the prior hypercube H to the smallest coordinate aligned cube containing the feasible set.

9.3 A Bounding Polytope of the Feasible Set

Polyhedra are geometric solids with flat faces and straight edges [20]. Definitions of polyhedra vary, but we use the definition of Boyd and Vandenberghe [14] by requiring such shapes to be convex as well. There are many good reasons to work with them, not least of which is that the feasible sets of linear programs are polyhedra. Therefore, it might be beneficial to construct a polyhedra that tightly bounds a feasible set. This can be easily done using Data Collaboration techniques.

One way to specify a polyhedron is as the intersection of half-planes. Given vectors $\mathbf{a}_i \in \mathbb{R}^n$ and scalars b_i (say $i = 1, \ldots, m$), the resulting polyhedron $\mathcal{P}$ is written as

$$\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}_i^\mathsf{T}\mathbf{x} \leq b_i, \text{ for } i = 1, \ldots, m\}. \tag{9.1}$$

If the polyhedron is bounded, it is called a polytope. Using response predictions techniques, we can find a polytope that bounds the feasible set.

The posterior hyperrectangle as described in the previous section is a polytope. Further constraints can be added for arbitrary directions $\mathbf{a}_i$. The value of the associated b_i is computed using a response prediction.

$$L_i = \min_{\mathbf{x}\in\mathcal{F}} \mathbf{a}_i^\mathsf{T}\mathbf{x}, \quad R_i = \max_{\mathbf{x}\in\mathcal{F}} \mathbf{a}_i^\mathsf{T}\mathbf{x}. \tag{9.2}$$

The outer bounds $\underline{L}_i$ and $\overline{R}_i$ are used to provide halfplane bounds on the feasible set.

$$\mathbf{a}^\mathsf{T}\mathbf{x} \leq \overline{R}_i \tag{9.3}$$

$$-\mathbf{a}^\mathsf{T}\mathbf{x} \leq -\overline{L}_i. \tag{9.4}$$

These types of constraints combined with the posterior (or prior) parameter bounds form a polytope.

The choice of directions $\mathbf{a}_i$ is rather arbitrary without more knowledge on the shape of the feasible set. As more and more half-plane constraints are added, the polytope improves as an approximation to the feasible set. Specifically, in the limit of infinite half-plane constraints in all directions, the polytope converges to the smallest convex set containing the feasible set: the convex hull of the feasible set. Therefore, it makes sense to constrain the feasible set with as many half-planes as possible, taking into account the amount of time to perform a prediction (which varies by problem).

One possible choice of directions $\mathbf{a}_i$ are the principal components of a sample of feasible points. Such a sample is obtained using the gas dynamics sampling algorithm (Algorithm 9.1). The principal components of this sample provide a basic shape of the feasible set. In this case the most important directions are those associated with the smallest variance in the data as these directions will provide the tightest new constraints.

9.4 A Bounding Ellipsoid of the Feasible Set

An ellipsoid is a generalization of an ellipse to higher dimensions. In n dimensions, an ellipsoid can be written as the level set of a convex quadratic function in n dimensions. In this work, we use the term *ellipsoid* to refer to an ellipsoidal solid, an ellipse along with its interior. An ellipse $\mathcal{E}$ can be written

$$\mathcal{E} := \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x}^\mathsf{T}\mathbf{Q}\mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x} + c \leq 0\}, \tag{9.5}$$

where $\mathbf{Q}$ is a $n \times n$ positive semidefinite matrix, $\mathbf{b}$ is an $n \times 1$ vector, and c is a scalar.

Ellipsoids are nice in that they can be described with one convex quadratic constraint. These can be used in quadratic programming (QP). Combined with a convex objective, the convex constrained quadratic program is in P (the class of problems solvable in polynomial time) [99], which is much better than the nonconvex quadratically constrained quadratic problem (see Chapter 3).

A sensible approach to finding a good ellipsoidal approximation to the feasible set would be to find the minimum volume ellipsoid that contains the feasible set (known as the *Löwner-John ellipsoid* [14]). However, unless the feasible set is described by *convex* quadratic constraints, the formulation of this problem is not readily solved [14]. Therefore, we provide another method for finding a covering ellipsoid.

By sampling the feasible set using the method outlined in Section 9.1, it's possible to obtain a spread of points that are representative of the shape of the feasible set (assuming a connected feasible set). Principal component analysis (PCA) of the sample produces a set of ordered orthogonal basis vectors for the space, such that the first principal direction accounts for the most variability possible in the sample,

the direction accounts for the second most variability, and so on. The directions and variances provide an ellipsoidal shape, and the arithmetic mean of the sampling provide the ellipsoid's center.

Let $\overline{\mathbf{x}}$ be the arithmetic mean of the m samples of the feasible set. Subtract off the mean from the samples and stack the resulting vectors horizontally into an $n \times m$ matrix $\mathbf{X}$. Take the singular value decomposition

$$\mathbf{X} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^{\mathsf{T}} \\ \mathbf{V}_2^{\mathsf{T}} \end{bmatrix} \tag{9.6}$$

where $\mathbf{\Sigma}$ is positive definite and diagonal, $[\,\mathbf{u}_1\ \mathbf{u}_2\,]$ and $[\,\mathbf{v}_1\ \mathbf{v}_2\,]$ are unitary matrices, and the zero matrices are of appropriate size. Then the quadratic coefficients of our bounding ellipse will be contained in the matrix $\mathbf{Q} = \left(\mathbf{U}_1 \mathbf{\Sigma} \mathbf{U}_1^{\mathsf{T}}\right)^{-1}$. Since we are centering our ellipsoid at $\overline{\mathbf{x}}$, we need only find the constant term c in the quadratic representation.

The problem is most intuitively formed as an optimization with a set containment constraint.

$$\begin{aligned} c^\star := \underset{c}{\operatorname{argmin}}\, & c \\ \text{s.t. } & \{\mathbf{x} \,:\, \mathbf{x}^{\mathsf{T}}\mathbf{Q}\mathbf{x} - 2\overline{\mathbf{x}}^{\mathsf{T}}\mathbf{Q}\mathbf{x} \leq c\} \subseteq \mathcal{F}. \end{aligned} \tag{9.7}$$

This problem can be rewritten as an NQCQP,

$$c^\star = \max_{\mathbf{x} \in \mathcal{F}} \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right]^{\mathsf{T}} \begin{bmatrix} 0 & \overline{\mathbf{x}}^{\mathsf{T}}\mathbf{Q} \\ \mathbf{Q}\overline{\mathbf{x}} & \mathbf{Q} \end{bmatrix} \left[\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}\right], \tag{9.8}$$

of which the outer bound can be solved using the S-procedure as outlined in Section 3.1.1.

9.5 Convex Outer Approximation of a Nonconvex Quadratically Constrained Set

If the feasible set is made of nonconvex quadratic constraints, then we can approximate it using convex quadratic constraints. Let the feasible set be made of p constraints, written as

$$L_k \leq \mathbf{x}^{\mathrm{T}}\mathbf{Q}_k\mathbf{x} + \mathbf{b}_k^{\mathrm{T}}\mathbf{x} + c_k \leq R_k \tag{9.9}$$

where $k = 1, \ldots, p$, L_k and R_k are scalars, and $\mathbf{Q}_k$, $\mathbf{b}_k$, and c_k are the quadratic, linear, and constant coefficients, respectively.

The matrices $\mathbf{Q}_k$ can be decomposed into matrices with only nonnegative or nonpositive eigenvalues using an eigenvalue decomposition. Since each $\mathbf{Q}_k$ is assumed to

be symmetric we can write the eigenvalue decomposition as

$$\mathbf{Q}_k = \begin{bmatrix}\mathbf{V}_{\text{pos}}\mathbf{V}_{\text{neg}}\mathbf{V}_0\end{bmatrix} \begin{bmatrix}\mathbf{\Lambda}_{\text{pos}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}_{\text{neg}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0}\end{bmatrix} \begin{bmatrix}\mathbf{V}_{\text{pos}}\mathbf{V}_{\text{neg}}\mathbf{V}_0\end{bmatrix}^{\mathsf{T}} \tag{9.10}$$

where $\mathbf{\Lambda}_{\text{pos}}$ is the diagonal matrix of the positive eigenvalues, $\mathbf{\Lambda}_{\text{neg}}$ is the diagonal matrix of the negative eigenvalues, and $\mathbf{V}_{\text{pos}}$, $\mathbf{V}_{\text{neg}}$, and $\mathbf{V}_0$ are the corresponding matrices of eigenvectors. By defining the matrices $\mathbf{Q}_{k,\text{pos}} := \mathbf{V}_{\text{pos}}\mathbf{\Lambda}_{\text{pos}}\mathbf{V}_{\text{pos}}^{\mathsf{T}}$ and $\mathbf{Q}_{k,\text{neg}} := \mathbf{V}_{\text{neg}}\mathbf{\Lambda}_{\text{neg}}\mathbf{V}_{\text{neg}}^{\mathsf{T}}$, the matrix $\mathbf{Q}_k$ can be decomposed as

$$\mathbf{Q}_k = \mathbf{Q}_{k,\text{pos}} + \mathbf{Q}_{k,\text{neg}}, \tag{9.11}$$

where $\mathbf{Q}_{k,\text{pos}}$ is positive semidefinite and $\mathbf{Q}_{k,\text{neg}}$ is negative semidefinite.

For each value of k we solve outer bounds of the following two optimization problems using the S-procedure.

$$\gamma_k := \max_{\mathbf{x}\in\mathcal{F}} \ \mathbf{x}^{\mathrm{T}}\mathbf{Q}_{k,\text{pos}}\mathbf{x} + \mathbf{b}_k^{\mathrm{T}}\mathbf{x} + c_k\omega_k := \min_{\mathbf{x}\in\mathcal{F}} \ \mathbf{x}^{\mathrm{T}}\mathbf{Q}_{k,\text{neg}}\mathbf{x} + \mathbf{b}_k^{\mathrm{T}}\mathbf{x} + c_k \tag{9.12}$$

where, again, $\mathcal{F}$ is the feasible set.

The convex feasible set approximation is created by using the prior parameter bound constraints (which are convex) and further constraining the convex and concave quadratic equations by γ_k and ω_k, respectively.

$$\mathbf{x}^{\mathrm{T}}\mathbf{Q}_{k,\text{pos}}\mathbf{x} + \mathbf{b}_k^{\mathrm{T}}\mathbf{x} + c_k \leq \gamma_k \tag{9.13}$$

$$\mathbf{x}^{\mathrm{T}}\mathbf{Q}_{k,\text{neg}}\mathbf{x} + \mathbf{b}_k^{\mathrm{T}}\mathbf{x} + c_k \geq \omega_k \tag{9.14}$$

These constraints are convex. This feasible set approximation contains the actual feasible set and hence is a conservative approximation.

9.6 Depth of a Point in a Nonconvex Quadratically Constrained Set

The depth of a point $\overline{\mathbf{x}}$ in a set is the smallest Euclidean distance from $\overline{\mathbf{x}}$ to a point on the boundary of the set. This is equivalent to the radius of the largest 2-norm ball centered at $\overline{\mathbf{x}}$ that is a subset of the set. If the point $\overline{\mathbf{x}}$ is outside of the boundary of the set, this definition of depth is equivalent to distance as discussed in Section 9.7. We will assume that the depth property is only defined for elements of the set and its boundary.

For a point $\overline{\mathbf{x}}$ and a set $\mathcal{F}$ such that $\overline{\mathbf{x}} \in \mathcal{F}$, let $\text{depth}(\overline{\mathbf{x}}, \mathcal{F})$ notate the depth of $\overline{\mathbf{x}}$ in $\mathcal{F}$. Specifically, it shall be defined as

$$\begin{aligned}\text{depth}(\overline{\mathbf{x}}, \mathcal{F}) := \max_{r} \ & r \\ \text{s.t. } & \{x \,:\, \|x - \overline{x}\|_2 \leq r\} \subseteq \mathcal{F}\end{aligned} \tag{9.15}$$

Furthermore, if the set is defined by m nonconvex quadratic constraints, described by the $n+1 \times n+1$ coefficient matrices $\{\mathbf{Z}_i\}_{i=1}^m$ as before, the depth property can be written with semidefinite constraints. Using the S-procedure, equation (9.15) is rewritten as

$$\begin{aligned} \text{depth}(\overline{\mathbf{x}}, \mathcal{F}) = \max_{r,\boldsymbol{\lambda}} \; & r \\ \text{s.t.} \; & \begin{cases} \left[\begin{smallmatrix} \overline{\mathbf{x}}^T\overline{\mathbf{x}} & \overline{\mathbf{x}}^T \\ \overline{\mathbf{x}} & \mathbf{I} \end{smallmatrix}\right] - \left[\begin{smallmatrix} r^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] - \lambda_i \mathbf{Z}_i \succeq 0, \; i = 1, \ldots, m. \\ \boldsymbol{\lambda} \geq 0. \end{cases} \end{aligned} \tag{9.16}$$

A semidefinite program is affine in all of its variables. Therefore the square of the depth can be solved with the SDP

$$\begin{aligned} \text{depth}^2(\overline{\mathbf{x}}, \mathcal{F}) = \max_{R,\boldsymbol{\lambda}} \; & R \\ \text{s.t.} \; & \begin{cases} \left[\begin{smallmatrix} \overline{\mathbf{x}}^T\overline{\mathbf{x}} & \overline{\mathbf{x}}^T \\ \overline{\mathbf{x}} & \mathbf{I} \end{smallmatrix}\right] - \left[\begin{smallmatrix} R & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] - \lambda_i \mathbf{Z}_i \succeq 0, \; i = 1, \ldots, m \\ \boldsymbol{\lambda} \geq 0, \end{cases} \end{aligned} \tag{9.17}$$

where R is used to replace r^2. However if the number of constraints used to describe the set is large, there are a lot of semi-definite constraints which implies longer computation times. For a point in the GRI-Mech 3.0 dataset [106], with total 356 constraints, the computation can take close to 5 minutes. However, this problem can be written as the minimum of m separate problems as

$$\begin{aligned} \text{depth}^2(\overline{\mathbf{x}}, \mathcal{F}) = \min_{i \in \{1,\ldots,m\}} \max_{R,\boldsymbol{\lambda}} \; & R \\ \text{s.t.} \; & \begin{cases} \left[\begin{smallmatrix} \overline{\mathbf{x}}^T\overline{\mathbf{x}} & \overline{\mathbf{x}}^T \\ \overline{\mathbf{x}} & \mathbf{I} \end{smallmatrix}\right] - \left[\begin{smallmatrix} R & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] - \lambda \mathbf{Z}_i \succeq 0 \\ \lambda \geq 0. \end{cases} \end{aligned} \tag{9.18}$$

Furthermore, if there are constraints that are simply the bounds on individual coordinates $\alpha_i \leq x_i \leq \beta_i$ (e.g. the prior bounds $\mathcal{H}$ in the Data Collaboration framework), then they can be handled simply by an upper bound on R. Let $r_{\max}$ be the minimum of all $\beta_i - \overline{x}_i$ and all $\overline{x}_i - \alpha_i$, then $R \leq r_{\max}^2$.

$$\begin{aligned} \text{depth}^2(\overline{\mathbf{x}}, \mathcal{F}) = \min_{i \in \{1,\ldots,m\}} \max_{R,\boldsymbol{\lambda}} \; & R \\ \text{s.t.} \; & \begin{cases} \left[\begin{smallmatrix} \overline{\mathbf{x}}^T\overline{\mathbf{x}} & \overline{\mathbf{x}}^T \\ \overline{\mathbf{x}} & \mathbf{I} \end{smallmatrix}\right] - \left[\begin{smallmatrix} R & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] - \lambda \mathbf{Z}_i \succeq 0 \\ \lambda \geq 0 \\ R \leq r_{\max}^2. \end{cases} \end{aligned} \tag{9.19}$$

The concept of depth is tied to the concept of parameter optimization and finding representative parameter values. The goal of parameter optimization is to find parameter values that cause the value of models to deviate the least from their respective experimental data. In Chapter 5 we described a method that attempts to find a feasible point with the minimal number of deviations of the parameters from their nominal values. Another notion of a representative parameter is a feasible point that is far from all constraint boundaries. This could be seen as finding the point with maximal depth. This is the same as allowing $\overline{\mathbf{x}}$ to be an optimization variable in equation (9.15). However, in this case, the formulation in equation (9.17) is no longer an SDP. Nonetheless, depth, as presented in this section, can be used to compare two feasible representative points.

9.7 Distance of a Point to a Nonconvex Quadratically Constrained Set

A common definition of the distance between a point and a set is the infimum of the distance between the point and elements of the set. If the set is closed, the distance becomes a minimum. By definition any element of the set (or any boundary point) has a distance of zero to the set.

For a point $\overline{\mathbf{x}}$ and a set $\mathcal{F}$, the distance between $\overline{\mathbf{x}}$ and $\mathcal{F}$ is denoted $\text{dist}(\overline{\mathbf{x}}, \mathcal{F})$ and computed with the optimization problem

$$\text{dist}(\overline{\mathbf{x}}, \mathcal{F}) := \min_{\mathbf{x} \in \mathcal{F}} \|\mathbf{x} - \overline{\mathbf{x}}\|_2. \tag{9.20}$$

If the set $\mathcal{F}$ is made of nonconvex quadratic constraints, then the square of the distance is a NQCQP and can be bounded using the techniques described in Chapter 3. Namely, the formulation becomes,

$$\text{dist}^2(\overline{\mathbf{x}}, \mathcal{F}) = \min_{\mathbf{x} \in \mathcal{F}} \|\mathbf{x} - \overline{\mathbf{x}}\|_2^2 = \min_{\mathbf{x} \in \mathcal{F}} \mathbf{x}^\mathsf{T}\mathbf{x} - 2\overline{\mathbf{x}}^\mathsf{T}\mathbf{x} + \overline{\mathbf{x}}^\mathsf{T}\overline{\mathbf{x}}. \tag{9.21}$$

For example, take the GRI-Mech dataset (Section 2.6). The optimized parameter values from the last two GRI-Mech releases (2.11 and 3.0) are not contained in the feasible set. The upper and lower bounds on the distance between the GRI-Mech 2.11 nominal vector and the feasible set is [5.03, 5.05]. The bounds on the distance for the GRI-Mech 3.0 nominal values is [5.92, 6.00], which is a little further away. This is a little surprising at first because the GRI-Mech 3.0 nominal value was optimized to minimize the error between the models and data. However, this is possible because when optimizing for GRI-Mech 3.0, researchers allowed only 31 of the 102 parameters to deviate from their nominal. One of their goals was to allow only a small number of parameters to deviate. The distance presented here uses the Euclidean norm and therefore all parameters may deviate between the nominal to the closest feasible point. In Section 5.5, this example is presented where the cardinality of the difference between the nominal and a feasible point is minimized instead of the 2-norm of the difference.

Chapter 10

Exploring the Feasible Set

Many properties of high dimensional feasible sets are very difficult to calculate. Estimations of the volume of a set, even if it is convex, are either poor estimates, or use inefficient algorithms [11]. Even calculating the volume of a polytope is difficult [27]. Discovering whether a set is convex or not is difficult, because even nonconvex constraints can result in a convex set. Without knowing much about a feasible set's properties, it would seem that quantifying the uncertainty in a model prediction might be difficult. However, Data Collaboration methods (surrogate modeling and constrained optimization) are able to do just that. In this chapter, we explore several ways to examine the size and shape of a feasible set. Several approximations were constructed of the feasible set and we explore how the approximations affect uncertainty propagation.

10.1 Rejection Sampling

Consider a set A that contains the feasible set, and is easy to sample. Sample points in A and reject any samples that are not in the feasible set. This is called a *rejection sampling* of the feasible set. The percentage of samples that are not rejected is an approximation to the ratio of the feasible set volume to the volume of A. If very few samples are rejected, we expect A to be a good approximation of the feasible set.

Rejection sampling can be used to examine the various feasible set approximations presented in Chapter 9. Specifically we compute several approximations for the GRI-Mech 3.0 feasible set described in Section 2.6. As a basis for comparison, 750 million points were sampled uniformly over the prior bounds hypercube, $\mathcal{H}$. Using the gas dynamics sampling algorithm described in Section 9.1, 750 million points were sampled for each of several other feasible set approximations. A toy example in 2

dimensions is shown for the feasible set and each type of sampled approximation is shown in Figure 10.1.

The labels correspond to the following sets.

(a) **The Feasible Set**. In the case of the GRI-Mech 3.0 dataset, this is a set made of 76 double sided nonconvex constraints and a double sided constraint on each the 102 parameters.

(b) **A Truncated Rotated Hyperrectangle**. Outer bounds of the feasible set were found in the principal component directions and were obtained from a sample of the feasible set (as in Section 9.4). This rotated hypercube is then further constrained (truncated) by the prior bounds $\mathcal{H}$, resulting in a bounding polytope as described in Section 9.3.

(c) **An Ellipse**. A bounding ellipse was found using the method described in Section 9.4.

(d) **A Truncated Ellipse**. This approximation was constructed by further constraining the ellipse in (c) by the prior bound $\mathcal{H}$.

(e) **A Convex Quadratic Approximation**. Using the method in Section 9.5, a convex outer bound of the feasible set was constructed.

(f) **Prior Bounds Hyperrectangle**. The prior information on the parameters is used without constraints from experiment data to construct this approximation.

In each approximation case, of all the sampled points, not a single point was found to be in the feasible set.

The GRI-Mech 3.0 problem is in 102 dimensions. In high dimensions these approximations can be quite poor because of the large amount of volume along the edges of the set. At the end of this chapter, in Section 10.6 we will provide further evidence that these approximations are poor by using them for Data Collaboration type questions.

10.2 Furthest Two Points in the Feasible Set

The prior bounds hyperrectangle for the GRI-Mech 3.0 dataset, $\mathcal{H}$, has been normalized to the hypercube $[-1,\, 1]^{102}$. Two opposite corners in $\mathcal{H}$ have a Euclidean (2-norm) distance of $2\sqrt{102} \approx 20.20$ between them. This is the furthest straight distance possible in $\mathcal{H}$. The smallest coordinate aligned hyperrectangle that contains the feasible set (as found in Section 9.2) has diagonal length of 19.37. By comparison, the longest distance between two points in the unit ball ($\{x : \|x\|_2 \leq 1\}$) is 2.

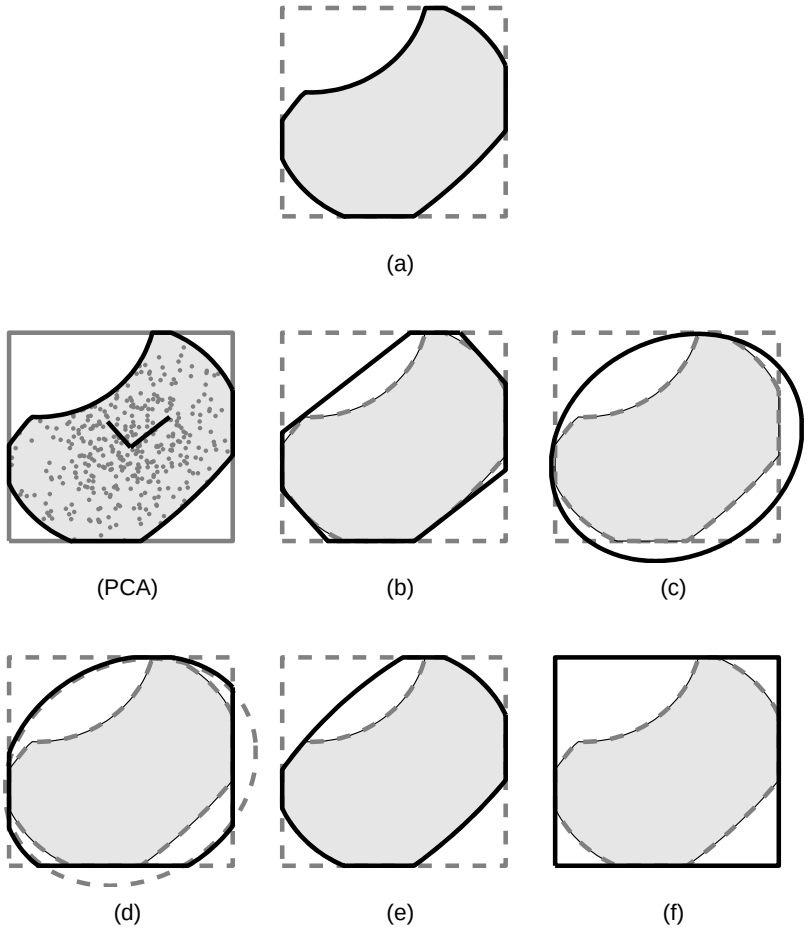

Figure 10.1: Various approximations to a toy feasible set in 2 dimensions. In each case the relevant set is shown with a bold outline, the actual feasible set is shaded, and the prior bounds are shown as a square. Approximation formulations are described throughout the text. See text for set descriptions. Subfigure (PCA) shows the principal component analysis of a feasible set sample resulting in the two directions shown.

Using `fmincon`, one of MATLAB's constrained optimization solvers, we were able to find two points in the GRI-Mech 3.0 feasible set that are a distance of 18.72 apart. This implies that the feasible set extends into corners of the hypercube, and is much longer than the unit ball in at least one direction. This would also appear to imply that the feasible set is similar in size to $\mathcal{H}$. We will show in Section 10.1 that this is not the case.

10.3 Testing for Convexity

A set K in $\mathbb{R}^n$ is called *convex* if $\theta\mathbf{x}+(1-\theta)\mathbf{y} \in K$ for every pair of points $\mathbf{x}$ and $\mathbf{y}$ in K and every scalar $\theta \in [0, 1]$. In words, a set is convex if the line segment between every pair of points in the set is also in the set. In general, the Data Collaboration feasible set is made of nonconvex constraints, and is therefore likely to be nonconvex (although this is not necessarily the case).

In Section 10.1, a convex approximation of the GRI-Mech 3.0 feasible set was shown to be much larger than the set itself. If the feasible set were constructed from convex constraints this approximation would have been exact, therefore the feasible set is possibly nonconvex. To explore the degree of nonconvexity of the GRI-Mech 3.0 feasible set, we picked random pairs of points in the set and looked for a point on the line segment that is not contained in the feasible set. After examining 70,000 pairs of feasible points, not a single connecting line that left the feasible set was found. Also the line segment between the two farthest apart feasible points found in Section 10.2 is completely contained in the feasible set. While this search is by no means a proof of convexity, it does seem to imply some sort of "roundness".

Rejection sampling showed that a convex superset is much bigger than the feasible set, yet every pair of points sampled defined line segments contained in the feasible set. Perhaps the nonconvexities are small ripples along the feasible set boundary. Yet in high dimensions, most of the volume is near the boundaries. This idea will be explored further in the next section.

10.4 Average Depth of the Feasible Set

Section 9.6 describes a method for finding the depth of a parameter vector in the feasible set. Section 9.1 demonstrated a method of sampling the feasible set. Using these tools, we computed the depth of 50,000 sampled feasible points. The largest depth found is 0.0021. The sample mean depth is 9.11×10^{-4}. This seems to imply that most of the feasible points are about half as far from the boundary as the deepest point found.

10.5 Comparison of a Feasible Set to the Unit Hypersphere and Prior Knowledge Bounds

So far, this chapter has explored several properties of the GRI-Mech 3.0 feasible set. To further this exploration, we compare the feasible set $\mathcal{F}$ to the prior bounds hypercube $\mathcal{H}$ (the unit ∞-norm ball for the GRI-Mech 3.0 dataset) and the unit ℓ_2-norm ball, defined as,

$$\mathcal{B} := \{\mathbf{x} \in \mathbb{R}^n \ : \ \|\mathbf{x}\|_2 \leq 1\}. \tag{10.1}$$

The results of this exploration are summarized in Table 10.1.

Table 10.1: Property comparison of the GRI-Mech 3.0 feasible set $\mathcal{F}$ to the prior bounds hypercube $\mathcal{H}$ and the unit ℓ_2-norm ball $\mathcal{B}$. † denotes a value that was computed with MATLAB's `fmincon`. * denotes a value that was computed via sampling of the set.

Property		$\mathcal{F}$	$\mathcal{H}$	$\mathcal{B}$
Convex?		No?	Yes	Yes
Convex by sampling test?*		Yes	Yes	Yes
Maximum distance b/t points		$18.72^\dagger$	20.20	2
Mean distance		4.97^*	8.23^{*1}	1.40 [2]
Standard deviation of distance		0.58^*	0.48^*	0.07 [2]
Contains origin?		No	Yes	Yes
Contains GRI-Mech 3.0 nominal?		No	Yes	No
Contains GRI-Mech 2.11 nominal?		No	Yes	No
Maximum depth		0.0021^*	1	1
Mean depth		$9.11 \times 10^{-4*}$	9.71×10^{-3}	9.71×10^{-3}
Depth standard deviation		$5.73 \times 10^{-4*}$	9.83×10^{-2}	9.83×10^{-2}
Volume		?	5.07×10^{30}	1.46×10^{-41}
Rjct. smpl.*	Rotated Truncated Hyperrectangle	0%	100%	0%
	Ellipse	0%	0%	5.4×10^{-4}%
	Truncated Ellipse	0%	100%	4.57×10^{-4}%
	Convex Approx	0%	100%	100%
	Prior Bounds, $\mathcal{H}$	0%	100%	0%

Most of the properties of the feasible set are similar to those of the prior knowledge hyperrectangle $\mathcal{H}$. However, rejection sampling implies that it might be more similar in size to the unit Euclidean-norm ball, $\mathcal{B}$. While doing this type of exploratory analysis may seem appealing, for uncertainty quantification and propagation it is unnecessary. The next section demonstrates this.

[1] Using techniques from [4], we can hard bound the average distance between two points in $\mathcal{H}$ in the interval [4.12, 8.25]. Exact solutions are known for up to 3-dimensions [90] but have not been generalized to n dimensions.

10.6 Predictions on Feasible Set Approximations

The Data Collaboration methods include the ability to predict the value of a model by looking for its maximum and minimum values over the feasible set of parameters. The methodology allows us to compute the prediction with the feasible set replaced by one of the approximations discussed in Chapter 9.

Take for example, the GRI-Mech 3.0 target F5, a laminar flame speed of a methane-air mixture [106]. As mentioned in Section 4.5, this target was excluded from the constraints of the feasible set. Instead, a prediction of target F5 is made using the rest of the GRI-Mech 3.0 dataset. Using the Data Collaboration methods described in Chapter 2 results in a prediction interval with outer bounds of $[33.65, 35.81]$ cm/sec, and inner bounds of [34.07, 35.61] cm/sec.

Predictions of target F5 are also made over the feasible set approximations mentioned in Chapter 9 and outlined in Section 10.1. Specifically, we calculated the response prediction using the truncated rotated hyperrectangle, the bounding ellipse, the truncated bounding ellipse, and the convex quadratic approximations as well as the prior bounds only. Results are shown in Figure 10.2. The prediction over the real feasible set gives the tightest prediction bounds. Furthermore the simplest approximation of the feasible set (the bounding ellipse) provides the largest prediction interval.

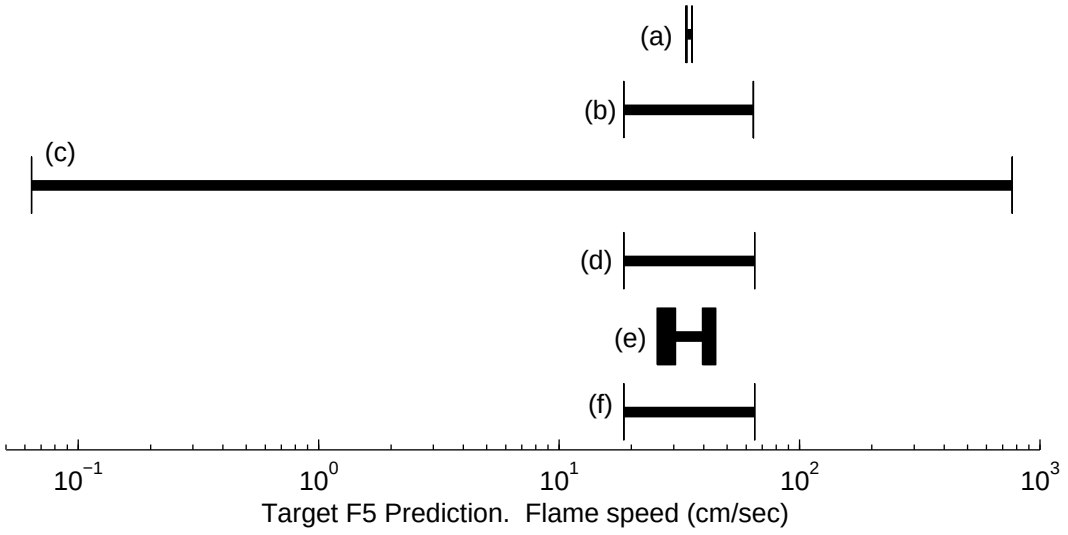

Figure 10.2: Prediction of GRI-Mech 3.0 target F5 using the actual feasible set and various approximations. The width of the box on each bound represents the differences in inner and outer bound predictions. The various predictions correspond to the type of constraints in Figure 10.1, namely (a) the actual feasible set, (b) a rotated and truncated hyperrectangle, (c) an ellipse, (d) a truncated ellipse, (e) convex quadratic approximation, and (f) the prior bound hypercube.

Chapter 11

Summary and Conclusions

This dissertation has explored some techniques for quantifying the uncertainty in collections of experimental data and simulation models. The techniques are mostly geared toward formulating various objectives as constrained optimization problems over the set of system parameters. These objectives include consistency of models and experiment data, prediction of unmeasured observables, and determining a representative parameter vector. The contributions provided in this work are expansions and improvements on previously published work, as well as some new techniques for surrogate fitting.

Data Collaboration is a framework presented in our previous work that combines both experiment prediction models and experimental data to constrain a set of uncertain parameters. Typical means of quantifying uncertainty in parameters lead to distilling the information gathered from experiments down to easy-to-describe-and-utilize results. Data Collaboration takes a different approach by using the collective experiment data (as opposed to results) from several sources to tightly constrain the parameters. Parameter constraints naturally fit in an optimization context. From this perspective, the consistency of a collection of data and models is simply a feasibility test of the constrained set. Prediction of an unmeasured observable is presented as the maximum and minimum values a model of the observable can take over the set of feasible parameters. Furthermore, when the models for the observables as functions of the uncertain parameters have quadratic forms, techniques from convex optimization and control theory are used to outer bound the optimization results, thereby providing a provable conservative bound on the objectives.

The basis of the optimization techniques used are those with nonconvex quadratically constrained programs. A short review of NQCQP techniques for inner and outer bound calculations is provided in Chapter 3. The outer bound optimizations — achieved with the S-procedure and its dual formulation — provide sensitivity information via Lagrange multipliers. The Lagrange multipliers are the partial derivatives

of the outer bound optimal value to the bounds on the constraints. These can then easily be transformed to get the partial derivatives of the optimal outer bounds to the experiment and parameter uncertainties. The sensitivities can provide a general measure of experiment impact, and also help create a linearization for the map from uncertainty to optimal values.

In many situations it is advantageous to have a vector of parameter values that are representative of the current knowledge of these parameters. Often this is the vector of nominal literature values for the parameters. It can also be the result of optimizing the parameters to improve correlation with experimental data. We furthered this idea by applying the concepts of compressed sensing to reduce the number of parameters required to deviate from their nominal values to achieve feasibility. The technique uses the one-norm as a heuristic replacement for the cardinality function. This can be rewritten as an NQCQP, and therefore outer bounds are computed quickly and with associated sensitivities.

To facilitate the generation of quadratic surrogate models for the observable models we created techniques for searching for an active subspace of the parameters. For complex simulation models with many parameters and long simulation times, quadratic surrogate modeling can be the limiting factor. By searching for a hopefully low-dimensional active subspace we can significantly reduce the amount of simulations required to build these surrogate models. An active subspace discovery algorithm was developed that exhibited the ability to compute a low-dimensional active subspace (given its existence) in a relatively low number of simulations. This technique is particularly useful when gradient calculations are available with each simulation and also when several of the observables of interest come from the same simulation.

Once an active subspace is found we have developed a procedure for generating experiment designs on the subspace that adhere to the original parameter bounds. These designs choose simulation locations in the parameter space such that they are spread out along the active subspace bases. We presented two methods for densely sampling the subspace, and three different optimization problems for selecting the design from the sample. There are pros and cons to each sampling method with some improvements made by sampling along the polytope boundaries. The three optimization problems yielded similar results in fitting errors, but were very different in computation time — the E-optimal design computation being the fastest. The optimizations could choose any desired number of design points from the samples; however, the design points tended to clump together after a certain number were chosen. Several suggestions are made to expand the design size beyond this number.

Several examples of using these active subspace method for surrogate generation yielded very positive results. For the GRI-Mech 3.0 target CH3.C1a, the techniques were able to achieve similar fitting errors to a fit generated over all parameters with only about 20% of the number of evaluations and computation time. An example in systems biology proved to benefit even more from the technique. In this case, by adding linear or pure quadratic terms in the direction orthogonal to the active subspace, even less fitting error was sometimes achieved for the cost of not many more

simulations, particularly with the maximum error in this case. The time to compute the consistency measure was also significantly reduced by using these techniques.

Even with good surrogate models that are fast to evaluate, if their domains are in high dimensions, the feasible set characteristics are difficult to discover and describe. The GRI- Mech 3.0 dataset appears to be very small as compared to the prior domain hypercube, or any of several feasible set approximations. A line segment that is partially out of the GRI-Mech 3.0 dataset, but has endpoints in the set, could not be found, implying a relatively convex feasible set. Yet, rejection sampling implied that the feasible set is much smaller than a convex approximation. Data Collaboration techniques allow us to make model response predictions on a feasible set without knowing its shape or complexity. Creating approximations to the feasible set that are convex and possibly easier to describe (e.g. linear or few constraints) adds little benefit to the analysis. Response predictions on the feasible set approximations presented shown in this document are not as precise and informative as a prediction on the actual feasible set or those using surrogate models.

Bibliography

[1] H. Agarwal, J. E. Renaud, E. L. Preston, and D. Padmanabhan. Uncertainty quantification using evidence theory in multidisciplinary design optimization. *Reliability Engineering & System Safety*, 85:281–294, 2004.

[2] V. S. Alagar. The distribution of the distance between random points. *Journal of Applied Probability*, 13(3):558–566, 1976.

[3] Alliance for Cellular Signaling. Homepage. http://www.signaling-gateway.org.

[4] R. S. Anderssen, R. P. Brent, D. J. Daley, and P. A. P. Moran. Concerning $\int_0^1 \dots \int_0^1 (x_1^2 + \dots + x_k^2)^{1/2} dx_1 \dots dx_k$ and a taylor series method. *SIAM Journal of Applied Mathematics*, 30(1):22–30, 1976.

[5] S. K. Au and J. L. Beck. Important sampling in high dimensions. *Structural Safety*, 25:139–163, 2003.

[6] H.-R. Bae, R. V. Grandhi, and R. A. Canfield. Epistemic uncertainty quantification techniques including evidence theory for large-scale structures. *Computers and Structures*, 82:1101–1112, 2004.

[7] J. S. Baehr. A priori and a posteriori. Internet Encyclopedia of Philosophy, 2006. Available at `http://www.iep.utm.edu/apriori/`.

[8] Z. Bai. Krylov subspace techniques for reduced-order modeling of large-scale dynamical systems. *Applied Numerical Mathematics*, 43:9–4, 2002.

[9] V. Balakrishnan, S. Boyd, and S. Balemi. Branch and bound algorithm for computing the minimum stability degree of parameter-depenent linear systems. *International Journal of Robust and Nonlinear Control*, 1(4):295–317, 1992.

[10] H.-W. Bandemer. *Mathematics of Uncertainty: Ideas, Methods, Application Problems*. Springer-Verlag, Berlin, Germany, 2006.

[11] I. Bárány and Z. Füredi. Computing the volume is difficult. *Discrete & Computational Geometry*, 2:319–326, 1987.

[12] V. Bier. Implications of the research on expert overconfidence and depedence. *Reliability Engineering & System Safety*, 85:321–329, 2004.

[13] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. Wiley, 1978.

[14] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[15] S. Burer and D. Vandenbussche. A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations. *Mathematical Programming*, 113(2):259–282, 2008.

[16] S. Burer and D. Vandenbussche. Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound. *Computational Optimization and Applications*, 43(2):181–195, 2009.

[17] E. Candes and T. Tao. Decoding by linear programming. *IEEE Trans. Inform. Theory*, 2004.

[18] A. Y. Chang, D. F. Davidson, M. DiRosa, R. Hanson, and C. Bowman. Shock tube experiments for development and validation of kinetic models of hydrocarbon oxidation. Poster at Proceedings of 25th Symposium (International) on Combustion, 1994.

[19] G. Chin, Jr. and C. S. Lansing. Capturing and supporting contexts for scientific data sharing via the biological sciences collaboratory. In *Proc. of the 2004 ACM conference on Computer supported cooperative work*, pages 409–418, 2004.

[20] F. Christophersen. Mathematical necessities. In *Optimal Control of Constrained Piecewise Affine Systems*, pages 3–19. Springer-Verlag, Berlin, Germany, 2007.

[21] S. Dasgupta, C. Papadimitriou, and U. Vazirani. *Algorithms*. McGraw-Hill, New York, NY, 2008.

[22] K. Derinkuyu and M. Ç. Pınar. On the S-procedure and some variants. *Mathematical Methods of Operation Research*, 64:55–77, 2006.

[23] B. DeVolder, J. Glimm, J. W. Grove, Y. Kang, Y. Lee, K. Pao, D. H. Sharp, and K. Ye. Uncertainty quantification for multiscale simulations. *Journal of Fluids Engineering*, 124:29–41, 2002.

[24] D. L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 2004.

[25] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 100, pages 5591–5596, 2003.

[26] K. J. Dowding, M. Pilch, and R. G. Hills. Model validation challenge problems: Thermal problem. http://www.isds.duke.edu/ fei/samsi/Readings/ThermalProblemDescrip.pdf, 2006.

[27] M. E. Dyer and A. M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM Journal of Computing*, 17(5):967–974, 1988.

[28] F. N. Egolfopoulos, P. Cho, and C. K. Law. Laminar flame speeds of methane-air mixtures under reduced and elevated pressures. *Combustion and Flame*, 76:375–391, June 1989.

[29] M. S. Eldred, A. A. Giunta, S. F. Wojtkiewicz, Jr., and T. G. Trucano. Formulations for surrogate-based optimization under uncertainty. In *9th AIAA/ISSMO Symposium and Exhibit on Multidisciplinary Analysis and Optimization*, 2002.

[30] A. F. Emery and A. V. Nenarokomov. Optimal experiment design. *Measurement Science and Technology*, 9:864–876, 1998.

[31] M. Fazel. *Matrix Rank Minimization with Applications*. PhD thesis, Stanford University, Stanford, CA, 2002.

[32] R. Feeley, M. Frenklach, M. Onsum, T. Russi, A. Arkin, and A. Packard. Model discrimination using data collaboration. *Journal of Physical Chemistry A*, 110(21):6803–6813, March 2006.

[33] R. Feeley, P. Seiler, A. Packard, and M. Frenklach. Consistency of a reaction dataset. *Journal of Physical Chemistry A*, 108(44):9573–9583, October 2004.

[34] R. P. Feeley. *Fighting the Curse of Dimensionality: A method for model validation and uncertainty propagation for complex simulation models*. PhD thesis, University of California, Berkeley, CA, 2008.

[35] T. Fetz and M. Oberguggenberger. Propagation of uncertainty through multivariate functions in the framework of sets of probability measures. *Reliability Engineering & System Safety*, 85:73–87, 2004.

[36] R. Fletcher. *Pratical Methods of Optimization*. Wiley, Chichester, New York, 1987.

[37] B. Flury. *Common Principal Components and Related Multivariate Models*. Wiley, second edition, 1988.

[38] E. Frazzoli, Z.-H. Mao, J.-H. Oh, and E. Feron. Resolution of conflicts involving many aircraft via semidefinite programming. *Journal of Guidance, Control and Dynamics*, 24(1):79–86, 2001.

[39] M. Frenklach. Modeling. In W. C. Gardiner, Jr., editor, *Combustion Chemistry*, chapter 7, pages 423–453. Springer-Verlag, New York, 1984.

[40] M. Frenklach. Transforming data into knowledge—process informatics for combustion chemistry. In *Proc. of the Combustion Institute*, pages 125–140, 2007.

[41] M. Frenklach. Personal communication, 2009.

[42] M. Frenklach, A. Packard, and P. Seiler. Prediction uncertainty from models and data. In *Proc. American Contorl Conference*, pages 4135–4140, 2002.

[43] M. Frenklach, A. Packard, P. Seiler, and R. Feeley. Collaborative data processing in developing predictive models of complex reaction systems. *International Journal of Chemical Kinetics*, 36(1):57–66, 2004.

[44] M. Frenklach, H. Wang, and M. J. Rabinowitz. Optimization and analysis of large chemical kinetic mechanisms using the solution mapping method—combustion of methane. *Prog. Energy Combust. Sci.*, 18:47–73, 1992.

[45] B. Ganapathysubramanian and N. Zabaras. Modelling diffusion in random heterogeneous media: data-driven models, stochastic collocation and the variational multi-scale method. *Journal of Computational Physics*, 226:326–353, 2007.

[46] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[47] D. Georgiev and E. Klavins. Model discrimination of polynomial systems via stochastic inputs. In *47th IEEE Conference on Decision and Control*, 2008.

[48] S. Georgiou, C. Koukouvinos, and J. Seberry. Hadamard matrices, orthogonal designs and construction algorithms. In *DESIGNS 2002: Further computational and constructive design theory*, pages 133–205. Kluwer, Norwell, Massachusetts, 2003.

[49] G. Goto, T. Sato, M. Nakajima, and T. Sukemura. A 5454-b regularly structured tree multiplier. *IEEE Journal of Solid-State Circuits*, 27(9):1229–1236, 1992.

[50] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. T. DeWitt, and G. Herber. Scientific data management in the coming decade. *ACM SIGMOD Record*, 34(4):34–41, 2004.

[51] E. J. Grimme. *Krylov Projection Methods for Model Reduction*. PhD thesis, University of Illinois, Urbana Champaign, IL, 1997.

[52] F. He and A. G. Marshall. Weighted quasi-newton and variable-order, variable-step adams algorithm for determining site-specific reaction rate constants. *J. Phys. Chem. A*, 104:562–567, 2000.

[53] T. Just. Private communication with GRI-Mech researchers. See http://me.berkeley.edu/gri_mech/version30/targets30/text30.html., 1994.

[54] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, and V. Gurvich. Generating all verticies of a polyhedron is hard. *Discrete and Computational Geometry*, 39:174–190, 2008.

[55] V. Klee. On the complexity of d-dimensional voronoi diagrams. *Archiv der Mathematik*, 34(1):75–80, 1979.

[56] G. J. Klir. Generalized information theory: aims, results, and open problems. *Reliability Engineering & System Safety*, 85:21–38, 2004.

[57] I. O. Kozine and L. V. Utkin. An approach to combining unreliable pieces of evidence and their propagation in a system response analysis. *Reliability Engineering & System Safety*, 85:103–112, 2004.

[58] D. Kurowicka and R. Cooke. *Uncertainty Analysis with High Dimensional Dependence Modelling*. Wiley, West Sussex, England, 2006.

[59] S. Lall, J. E. Marsden, and S. Glavaski. A subspace approach to balanced truncation for model reduction of nonlinear control systems. *International Journal of Robust and Nonlinear Control*, 12(6):519–535, 2002.

[60] L. H. Lee and K. Poolla. Statistical validation for uncertainty models. In *Feedback Control, Nonlinear Systems, and Complexity*, volume 202/1995, pages 131–149. Springer, Berlin / Heidelberg, 1995.

[61] P. M. Lee. *Bayesian Statistics*. Wiley, New York, NY, USA, 2004.

[62] G. Lemon, W. G. Gibson, and M. R. Bennett. Metabotropic receptor activation, desensitization and sequestration-I: modelling calcium and inositol 1,4,5-trisphosphate dynamics following receptor activation. *Journal of Theoretical Biology*, 223(1):93–111, 2003.

[63] D. Lindley. On a measure of the information provided by an experiment. *Annals of Mathematical Statistics*, 27(4):986–1005, 1956.

[64] B. Liu. *Uncertainty Theory*. Springer-Verlag, Berlin, Germany, 2nd edition, 2007.

[65] Z. Lu and D. Zhang. A comparative study on uncertainty quantification for flow in randomly heterogeneous media using Monte Carlo simulations and conventional and KL-based moment-equation approaches. *SIAM Journal on Scientific Computing*, 26(2):558–577, 2005.

[66] G. Marsaglia. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics*, 43(2):645–646, 1972.

[67] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[68] I. McLean, D. Smith, and S. Taylor. The use of CO/H2 burning velocities to examine the rate of the CO-OH reaction. In *Proc. Combust. Inst.*, volume 25, page 749, 1994.

[69] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge, Cambridge, UK, 2005.

[70] D. C. Montgomery. *Design and Analysis of Experiments*. Wiley, second edition, 1984.

[71] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[72] S. R. Morrissey. 'Cyber-enabled' chemistry. *Chemical & Engineering News*, pages 28–30, October 2005.

[73] M. E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2(4):19–20, 1959.

[74] J. D. Myers, T. C. Allison, S. Bittner, B. Didier, M. Frenklach, W. H. Green, Jr., Y.-L. Ho, J. Hewson, W. Koegler, C. Lansing, D. Leahy, M. Lee, R. McCoy, M. Minkoff, S. Nijsure, G. von Laszewski, D. Montoya, L. Oluwole, C. Pancerella, R. Pinzon, W. Pitz, L. A. Rahn, B. Ruscic, K. Schuchard, E. Stephan, A. Wagner, T. Windus, and C. Yang. A collaborative informatics infrastructure for multi-scale science. *Cluster Computing*, 8(4):243–253, 2005.

[75] R. H. Myers and D. C. Montgomery. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley, 1995.

[76] M. New and M. Hulme. Representing uncertainty in climate change scenarios: a Monte-Carlo approach. *Integrated Assessment*, 1(3):1573–1545, 2000.

[77] W. L. Oberkampf, J. C. Helton, C. A. Joslyn, S. F. Wojtkiewicz, and S. Ferson. Challenge problems: Uncertainty in system response given uncertain parameters. *Reliability Engineering & System Safety*, 85:11–19, 2004.

[78] W. L. Oberkampf, T. G. Trucano, and C. Hirsch. Verification, validation, and predictive capability in computational engineering and physics. *Applied Mechanics Reviews*, 57(5):345–384, 2004.

[79] M. Oliver and R. Webster. Kriging: a method of interpolation for geographical information systems. *International Journal of Geographical Information Science*, 4(3):313–332, 1990.

[80] A. B. Owen. Latin supercube sampling for very high-dimensional simulations. *ACM Transactions on Modeling and Computer Simulation*, 8(1):71–102, 2003.

[81] A. K. Packard and S. S. Sastry. Solving rational matrix equations in the state space with applications to computer-aided control-system design. *Internation Journal of Control*, 43(1):65–90, 1986.

[82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, Mineola, NY, USA, 1982.

[83] P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[84] C. V. Phillips. Quantifying and reporting uncertainty from systematic errors. *Epidemiology*, 14(4):459–466, 2003.

[85] D. E. Post and L. G. Votta. Computational science demands a new paradigm. *Physics Today*, pages 35–41, January 2005.

[86] V. Preda and I. Chitescu. On constraint qualification in multiobjective optimization problems: Semidifferentiable case. *Journal of Optimization Theory and Applications*, 100(2):417–433, 1999.

[87] H. Rabitz and Ömer F. Alış. General foundations of high-dimensional model representations. *Journal of Mathematrical Chemistry*, 425:197–233, 1999.

[88] H. Rabitz, Ömer F. Alış, J. Shorter, and K. Shim. Efficient input-output model representations. *Computer Physics Communications*, 117:11–20, 1999.

[89] B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. In *Proc. Allerton Conference*, 2008. Updated version available at http://faculty.washington.edu/mfazel/.

[90] D. P. Robbins and T. S. Bolis. E 2629: Average distance between two points in a box. *The American Mathematical Monthly*, 85(4):277–278, 1978. Problem posed by Robbins and solved by Bolis.

[91] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[92] R. T. Rockafellar. Lagrange multipliers and optimality. *SIAM Review*, 35(2):183–238, June 1993.

[93] D. S. Roos. Bioinformatics—trying to swim in a sea of data. *Science*, 291(5507):1260–1261, 2001.

[94] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2320–2326, 2000.

[95] T. Russi, A. Packard, R. Feeley, and M. Frenklach. Sensitivity analysis of uncertainty in model prediction. *Journal of Physical Chemistry A*, 112(12):2579–2588, February 2008.

[96] S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, December 1974.

[97] A. Saltelli. What is sensitivity analysis? In A. Saltelli, K. Chan, and E. M. Scott, editors, *Sensitivity Analysis*, pages 3–13. John Wiley & Sons, Ltd, West Sussex, England, 2000.

[98] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. John Wiley & Sons, Ltd, West Sussex, England, 2004.

[99] S. Schmieta and F. Alizadeh. Associative and jordan algebras, and polynomial time interior-point algorithms for symmetric cones. *Mathematics of Operations Research*, 26(3):543–564, 2001.

[100] K. Schuchardt, J. Myers, and E. Stephan. A web-based data architecture for problem-solving environments: Application of distributed authoring and versioning to the extensible computational chemistry environment. *Cluster Computing*, 5(3):287–296, 2002.

[101] P. Seiler. NQCQP Notes. http://www.me.berkeley.edu/ pack/library/nqcqp.pdf, 2002.

[102] P. Seiler, M. Frenklach, A. Packard, and R. Feeley. Numerical approaches for collaborative data processing. *Optimization and Engineering*, 7(4):459–478, December 2006.

[103] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

[104] A. B. Singer, J. W. Taylor, P. I. Barton, and W. H. Green. Global dynamic optimization for parameter estimation in chemical kinetics. *Journal of Physical Chemistry A*, 110:971–976, 2006.

[105] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, Massachusetts, 2 edition, 2006.

[106] G. P. Smith, D. M. Golden, M. Frenklach, N. W. Moriarty, B. Eiteneer, M. Goldenberg, C. T. Bowman, R. K. Hanson, S. Song, W. C. Gardiner, Jr., V. V. Lissianski, and Z. Qin. GRI-Mech 3.0. http://www.me.berkeley.edu/gri_mech/.

[107] A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[108] J. Sturm. *Primal-dual interior point approach to semidefinite programming*. PhD thesis, Tinbergen Institute, Amsterdam, The Netherlands, 1997.

[109] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999.

[110] Q. Y. Tang. Personal communication, 2008.

[111] R. Taussig, R. Ranganathan, E. M. Ross, and A. G. Gilman. Overview of the alliance for cellular signaling. *Nature*, 420(12):703–706, 2002.

[112] P. W. Tuinenga. *SPICE: A Guide to Circuit Simulation and Analysis Using PSpice*. Prentice Hall, second edition, 1991.

[113] C. M. Vagelopoulos and F. N. Egolfopoulos. Direct experimental determination of laminar flame speeds. In *Proc. Combust. Inst.*, volume 27, pages 513–519, 1997.

[114] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[115] S. Vempala. Algorithmic convex geometry. Available at http://www.cc.gatech.edu/ vempala/acg/notes.pdf, 2008.

[116] C. Walter. Kryder's law. *Scientific American*, August 2005.

[117] S. Wojtkiewicz, M. Eldred, R. Field, Jr., A. Urbina, and J. Red-Horse. Uncertainty quantification in large computational engineering models. In *42nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, 2001.

[118] A. Yates. *Multivariate Exploratory Data Analysis: A perspective on Exploratory Factor Analysis*. State University of New York Press, 1987.

[119] T.-M. Yi, M. Fazel, X. Liu, T. Otitoju, J. Goncalves, A. Papachristodoulou, S. Prajna, and J. Doyle. Application of robust model validation using SOSTOOLS to the study of G-protein signaling in yeast. In *Foundations of Systems Biology in Engineering (FOSBE) 2005*, 2005.

[120] X. You. Personal communication, 2009.

[121] X. You, T. Russi, A. Packard, and M. Frenklach. Optimization of combustion kinetic models on a feasible set. In *Proc. Combust. Inst.*, volume 33, August 2010.

[122] Y. Zhao and K.-T. Fang. Orthogonal exact designs on a sphere, a spherical cap, or a spherical belt. *Journal of Statistical Planning and Inference*, 98:279–285, 2001.

Appendix A

Solving the NQCQP Lower Bound Problems Using SeDuMi

Throughout this paper nonconvex quadratically constrained quadratic programs are used to solve various problems. Often the lower bound problems derived using the S-procedure (3.2) or a rank relaxation (3.4) are used because of computational speed and Lagrange multiplier information as well as the lower bound itself. This appendix will express the general NQCQP (including equality constraints) as a conic linear program which many optimization solvers use. Section A.1 derivs the equations, and Section A.2 provides MATLAB code for use with the SeDuMi optimization software [109].

A.1 Conic Linear Program Formulation of NQCQP Lower Bound Problems

The general formulation for the NQCQP including equality constraints is

$$\begin{aligned} p^\star = \min_{\mathbf{x}\in\mathbb{R}^n} & \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Z}_0 \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \\ \text{s.t.} & \begin{cases} \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathbf{Z}_i \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] \leq 0, \ i = 1,\ldots,m, \\ \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right]^\mathsf{T} \mathfrak{Z}_j \left[\begin{smallmatrix}1\\ \mathbf{x}\end{smallmatrix}\right] = 0, \ j = 1,\ldots,q, \end{cases} \end{aligned} \tag{A.1}$$

where $\mathbf{Z}_i$ and $\mathfrak{Z}_j$, for $i = 0,\ldots,m$ and $j = 1,\ldots,q$, are $(n+1)\times(n+1)$ symmetric matrices. The equality constraints could be formulated as inequalities by constraining

the quadratics to be both greater than or equal to or less than or equal to zero. However, most solvers use more sophisticated techniques for handling equality constraints, so we use this formulation when equality constraints are involved.

The resulting lower bound from the S-procedure, $\underline{p}^\star_{\text{s}}$, and the lower bound from the rank relaxation, $\underline{p}^\star_{\text{r}}$, are written as

$$\begin{aligned} \underline{p}^\star_{\text{r}} = &\min_{\mathbf{Q}\succeq 0} \operatorname{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] \\ &\text{s.t.} \begin{cases} Q_{11} = 1, \\ \operatorname{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] \leq 0,\ i = 1,\ldots,m, \\ \operatorname{Tr}\left[\mathfrak{Z}_j\mathbf{Q}\right] = 0,\ j = 1,\ldots,q. \end{cases} \end{aligned} \tag{A.2}$$

$$\begin{aligned} \underline{p}^\star_{\text{s}} = &\max_{\boldsymbol{\lambda}\geq 0,\boldsymbol{\nu},\gamma} \gamma \\ &\text{s.t. } \mathbf{Z}_0 - \left[\begin{smallmatrix} \gamma & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_n \end{smallmatrix}\right] + \sum_{i=1}^{m} \lambda_i \mathbf{Z}_i + \sum_{j=1}^{q} \nu_j \mathfrak{Z}_j \succeq 0, \end{aligned} \tag{A.3}$$

The form of the linear conic program as used by the optimization package SeDuMi [109] is

$$\begin{aligned} p = &\min_{\mathbf{x}\in\mathbb{R}^N} \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ &\text{s.t.} \begin{cases} \mathbf{A}\mathbf{x} = \mathbf{b} \\ \mathbf{x} \succeq_K \mathbf{0} \end{cases} \end{aligned} \tag{A.4}$$

where $\mathbf{A}$ is a $M \times N$ matrix, $\mathbf{b}$ is a $N \times 1$ matrix, and $\mathbf{c}$ is $M \times 1$ matrix. Here the constraint $\mathbf{x} \succeq_K 0$ is a generalized inequality requiring $\mathbf{x}$ to be in the cone K [14]. The dual of this problem is

$$\begin{aligned} d = &\max_{\mathbf{y}\in\mathbb{R}^M} \mathbf{b}^{\mathsf{T}}\mathbf{y} \\ &\text{s.t. } \mathbf{c} - \mathbf{A}^{\mathsf{T}}\mathbf{y} \succeq_{K^*} 0 \end{aligned} \tag{A.5}$$

where K^* is the dual cone of K. We will now express the rank relaxation problem (A.2) as the primal linear conic program (A.4) and the S-procedure problem (A.3) as the dual linear conic program (A.5).

First we rewrite the rank relaxation formulation introducing the slack variables $\boldsymbol{\tau} \in \mathbb{R}^p$.

$$\begin{aligned} \underline{p}^\star_{\text{r}} = &\min_{\mathbf{Q}\succeq 0,\boldsymbol{\tau}\geq 0} \operatorname{Tr}\left[\mathbf{Z}_0\mathbf{Q}\right] \\ &\text{s.t.} \begin{cases} Q_{11} = 1, \\ \operatorname{Tr}\left[\mathbf{Z}_i\mathbf{Q}\right] + \tau_i = 0,\ i = 1,\ldots,m, \\ \operatorname{Tr}\left[\mathfrak{Z}_j\mathbf{Q}\right] = 0,\ j = 1,\ldots,p. \end{cases} \end{aligned} \tag{A.6}$$

Also, let $\operatorname{vec}(\cdot)$ be the function that maps an $(n+1)\times(n+1)$ matrix to an $(n+1)^2 \times 1$ vector as

$$\operatorname{vec}\left(\begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_{n+1} \end{bmatrix}\right) = \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{n+1} \end{bmatrix}. \tag{A.7}$$

Then the trace operations can be rewritten as

$$\mathrm{Tr}\left[\mathbf{X}\mathbf{Y}\right] = \mathrm{vec}(\mathbf{X})^{\mathsf{T}}\,\mathrm{vec}(\mathbf{Y}). \tag{A.8}$$

Now with this notation, the rank relaxation and S-procedure problems are rewritten in the linear conic program formulation.

$$\mathbf{A} := \left[\begin{array}{c|c} & -\mathrm{vec}(\mathbf{Z}_1)^{\mathsf{T}} \\ -\mathbf{I}_{m\times m} & \vdots \\ & -\mathrm{vec}(\mathbf{Z}_m)^{\mathsf{T}} \\ \hline & -\mathrm{vec}(\mathfrak{Z}_1)^{\mathsf{T}} \\ \mathbf{0}_{q\times m} & \vdots \\ & -\mathrm{vec}(\mathfrak{Z}_q)^{\mathsf{T}} \\ \hline \mathbf{0}_{1\times m} & 1 \quad \mathbf{0}_{1\times(n^2+2n)} \end{array}\right],$$

$$\mathbf{c} := \begin{bmatrix} \mathbf{0}_{m\times 1} \\ \mathrm{vec}(\mathbf{Z}_0) \end{bmatrix}, \qquad \mathbf{x} := \begin{bmatrix} \boldsymbol{\tau} \\ \mathrm{vec}(\mathbf{Q}) \end{bmatrix},$$

$$\mathbf{b} := \begin{bmatrix} \mathbf{0}_{m\times 1} \\ \mathbf{0}_{q\times 1} \\ 1 \end{bmatrix}, \qquad \mathbf{y} := \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\nu} \\ \gamma \end{bmatrix},$$

$$K := \left\{\mathbf{x} = \left[\begin{smallmatrix} \boldsymbol{\tau} \\ \mathrm{vec}(\mathbf{Q}) \end{smallmatrix}\right] : \boldsymbol{\tau} \geq 0,\, \mathbf{Q} \succeq 0\right\}.$$

These vectors $\mathbf{b}$, $\mathbf{c}$, $\mathbf{x}$, and $\mathbf{y}$ and matrix $\mathbf{A}$ provide the conversion of the lower bound problems into the linear conic programs as desired. This particular cone is self-dual, so $K^* = K$.

A.2 MATLAB code for NQCQP outer bounds using SeDuMi

The following MATLAB function solves the two NQCQP lower bound problems presented in this document using the freely available optimization package SeDuMi [109].

Algorithm A.1: MATLAB code for lower bounding of a NQCQP

```
function [lb1,lb2,Q,mults,info] = nqcqp_lowerbound(Z0,Zineq,Zeq,fid)
%Lower bound on the nonconvex quadratically constrained quadratic
```

```
%program
%
%  Provides lower bounds of the NQCQP
%           min_x  [1 x']*Z0*[1;x]
%           such that: [1 x']*Zineq{i}*[1;x], i=1,...,m
%                      [1 x']*Zeq{j}*[1;x],   j=1,...,p
%  Specifically it solves the program
%      lb1 = min_Q  trace(Z0*Q)
%            such that: trace(Zineq{i}*Q)<=0, i=1,...,m
%                       trace(Zeq{j}*Q)=0,    j=1,...,q
%                       Q(1,1)=1,
%                       Q is positive semidefinite
%  and its dual program
%      lb2 = max_{lambda,nu,gamma} gamma
%            such that: Z0 - [gamma 0; 0 0] + sum(lambda_i*Zineq{i}) +
%                          + sum(nu_j*Zeq{j}) is positive semidefinite
%                       lambda_i>=0, i=1,...,m
%  using SeDuMi.
%
%  Inputs:
%     Z0: an (n+1)-by-(n+1) symmetric matrix
%     Zineq: a m-by-1 cell array with each cell containing an
%         (n+1)-by-(n+1) symmetric matrix
%     Zeq: [optional] a q-by-1 cell array with each cell containing an
%         (n+1)-by-(n+1) symmetric matrix
%     fid: [optional, default=1] file identifier for SeDuMi progress
%         output.  If fid=1, the progress is printed to the screen.
%         If fid=0, the progress is supressed.  This can also be a
%         file identifier provided by the fopen command.
%
%  Outputs:
%     lb1: the value of the lower bound from the primal problem
%     lb2: the value of the lower bound from the dual problem
%     Q: the optimized primal decision variable, a (n+1)-by-(n+1)
%         matrix
%     mults: the optimized multipliers from the dual formulation, a
%         structure with fields
%         .ineq: a m-by-1 array (lambdas)
%         .eq: a q-by-1 array (nus)
%     info: structure provided by SeDuMi with information on the
%         optimization.  Most notably if info.pinf=1, then the primal
%         problem is infeasible, a if info.pinf=1, then the dual
%         problem is infeasible.  See the SeDuMi help for more
%         information.
```

```
ni = nargin;
no = nargout;

%check number of inputs/outputs
error(nargchk(2,4,ni,'struct'));
error(nargoutchk(0,5,no,'struct'));

%if no equality constraints provided, make an empty cell array
if ni<3
    Zeq = {};
    fid = 1;
elseif ni<4
    fid = 1;
end

%error check input classes
assert(isnumeric(Z0),'First input must be numeric array');
assert(iscell(Zineq)&&(isvector(Zineq) || isempty(Zineq)),...
    'Second input must be vector cell array');
assert(iscell(Zeq)&&(isvector(Zeq) || isempty(Zeq)),...
    'Third input must be vector cell array');
assert(isnumeric(fid)&&isscalar(fid)&&any(fid==[0 1 fopen('all')]),...
    'Fourth input must be 0, 1, or a valid file id provided by fopen')

%get some sizes
n = size(Z0,1)-1;
m = length(Zineq);
q = length(Zeq);

%error check sizes
assert(size(Z0,2)==n+1,...
    'First input needs to be a square symmetric matrix')
Z0 = 0.5*(Z0+Z0'); %make it symmetric in case of small numeric issues
for i=1:m %repeat check for each inequality constraint
    assert(all(size(Zineq{i})==[n+1 n+1]),...
        ['Each cell of second input must be the same size as the',...
        'first input'])
    Zineq{i} = 0.5*(Zineq{i}+Zineq{i}');
end
for j=1:q %repeat check for each equality constraint
    assert(all(size(Zeq{j})==[n+1 n+1]),...
        ['Each cell of second input must be the same size as the',...
        'first input'])
    Zeq{j} = 0.5*(Zeq{j}+Zeq{j}');
end
```

```
%build sparse matrix
A1 = [-speye(m); spalloc(q+1,m,0)]; %sparse matrix associated with
                                    %linear vector inequalities
A2 = zeros(m+q+1,(n+1)^2); %sparse matrix associated with
                           %linear matrix inequalities
for i=1:m
    A2(i,:) = -vec(Zineq{i})';  %vec is a function provided with the
                                %SeDuMi package
end
for j=1:q
    A2(m+j,:) = -vec(Zeq{j})';
end
A2(end,1) = 1;
A = sparse([A1 A2]);

%and vectors
b = spalloc(m+q+1,1,1);
b(end) = 1;
c = [spalloc(m,1,0); sparse(vec(Z0))];

%cone constraints
K.l = m; %first m elements of x are nonnegative
K.s = n+1; %next (n+1)^2 elements form a (n+1)-by-(n+1) matrix that is
           %positive semidefinite

%set progress output file id parameter
pars.fid = fid;

%sedumi call
[x,y,info] = sedumi(A,b,c,K,pars);

%create outputs
lb1 = c'*x;
lb2 = b'*y;
Q = mat(x(m+1:end)); %mat is a function by the SeDuMi package
mults.ineq = y(1:m);
mults.eq = y(m+(1:q));
```

Appendix B

Data Collaboration Toolbox for MATLAB

The Data Collaboration toolbox is a free, open-source software package for the MATLAB programming environment for non-stochastic uncertainty quantification using predictive models and experiment data. The software is available at http://collab-sci.sourceforge.net/

The software allows a user to enter experiment data with uncertainty along with a corresponding parameterized model. These combine to constrain a set of feasible parameter values that match the data and model. Questions about this feasible set of parameters, such as whether it is nonempty (consistency) and what the range of values a given predictive model can take over the set (response prediction), are posed as constrained optimization problems.

The techniques utilized in Data Collaboration (DC) are those described in several journal articles.

1. M. Frenklach, A. Packard, and P. Seiler. Prediction uncertainty from models and data. In Proc. American Control Conference, pages 4135–4140, 2002.

2. M. Frenklach, A. Packard, P. Seiler, and R. Feeley. Collaborative data processing in developing predictive models of complex reaction systems. International Journal of Chemical Kinetics, 36(1):57–66, 2004.

3. R. Feeley, P. Seiler, A. Packard, and M. Frenklach. Consistency of a reaction dataset. Journal of Physical Chemistry A, 108(44):9573–9583, October 2004.

4. R. Feeley, M. Frenklach, M. Onsum, T. Russi, A. Arkin, and A. Packard.

Model discrimination using data collaboration. Journal of Physical Chemistry A, 110(21):6803–6813, March 2006.

5. P. Seiler, M. Frenklach, A. Packard, and R. Feeley. Numerical approaches for collaborative data processing. Optimization and Engineering, 7(4):459–478, December 2006.

6. T. Russi, A. Packard, R. Feeley, and M. Frenklach. Sensitivity analysis of uncertainty in model prediction. Journal of Physical Chemistry A, 112(12):2579–2588, February 2008.

This document is designed to act both as a manual for the use of the toolbox, as well as providing some high-level description of the underlying algorithms. Each section describes either the creation of some objects or an algorithm. If you feel you would like to jump right in, we suggest you first read the requirements and downloading/installation instructions in Sections B.1 and B.2. After that skip to the examples in Section B.11.

B.1 Requirements

- **"Any" computer operating system**: DC has been tested on machines running Windows XP, Windows Vista x86 (32-bit), Windows Vista x64 (64-bit), and Ubuntu. In principle, it should run on any fairly new desktop or laptop computer that can run MATLAB.
- **MATLAB release 2008a or later**
- **Optimization toolbox for MATLAB**
- **SeDuMi optimization package**: SeDuMi is a free MATLAB toolbox for self-dual minimization. DC provides version 1.21, but you may use another version if you like. DC has been tested with SeDuMi versions 1.1 and 1.21.

B.2 Download and Installation

To download the software,

1. Visit the project home page at http://collab-sci.sourceforge.net/
2. Click the "Download Page" link.
3. Click the large green button that says "Download Now!"

4. Depending on your internet browser's preferences, you will either be prompted to choose a location, or the zip-file will be saved to your default download folder.

Once the download is complete, unzip the file to a convenient location (e.g. your default MATLAB working directory). To unzip a file you need a zip decompression program. For windows, we recommend the free utility, 7zip, available at http://www.7zip.com/. For linux/unix, there may already be a utility installed called gzip.

Inside the main folder are several subdirectories and a few files. One of these files is named "DCsetup.m," a MATLAB script for setting path directories and importing the main package. From the MATLAB command prompt run the script:

```
>> DCsetup
```

This will add the necessary directories to the current MATLAB path. It will also import the `+DClab` package. This setup script should be executed from the main DC directory each time you start MATLAB if you wish to use the DC toolbox. This script can be executed from a startup file.

All of the main DC methods and objects have the prefix `DClab.`, but this import will make it such that using this prefix will be unnecessary. If you wish to use DC objects and methods inside of a function or custom class, you either need to include the `DClab.` prefix, or use the command

```
import DClab.*
```

at the top of each file. This command is in the "DCsetup" file, but that only will import the package for the base workspace. For more information, see MATLAB documentation on packages. Throughout this documentation, we will not use the prefix for the sake of simplicity.

To test the directory setup, run the test script from the MATLAB command prompt

```
>> dctest
```

This will take a minute or two. One issue you may run into the first time you run the package is that the SeDuMi mex-files may need to be recompiled. This is only necessary one time.

If the test script gives an error regarding SeDuMi, change the current directory in MATLAB to the "SeDuMi_1_21" directory in the DC toolbox main directory. From the command prompt type

```
>> mex -setup
```

and select a c-compiler of your choice. For 32-bit operating systems, MATLAB ships with the free compiler lcc. If no compiler is shown, you may need to get a c-compiler. Please consult the MATLAB documentation for supported c-compilers. Once the compiler is chosen, type the following at the command prompt

```
>> install_sedumi
```

This will take a few minutes. Once complete, change back to the main DC directory and re-run dctest. Note: if the test script worked the first time, you should not have to reinstall sedumi.

B.3 Version 2.0

The software has changed significantly from the previous release (1.0) to this version (2.0). Most notably almost all of the object names has changed. Section B.12 provides a table to help you convert your own code to the names.

There are many changes to the underlying methods, and some ways that you can interact with the software (e.g. the ability to define multiple features from a single simulation file). All these are described in this document.

B.4 Dataset Formulation

The Data Collaboration Dataset is a collection of parameterized models, associated experiment observations, and constraints on the individual parameters. Let Y_e be some scalar observable. Model M_e predicts the value of the observable as a function of n unknown parameters. Each parameter x_i has a set of prior knowledge bounds α_i and β_i such that $\alpha_i \leq x_i \leq \beta_i$. Also associated with each observable is an experimental measurement d_e with lower and upper bound uncertainty l_e and u_e such that the value of the observable y_e is bounded as $l_e \leq y_e - d_e \leq u_e$. All this information collected together for m observables ($e = 1, \ldots, m$) is known as a dataset.

The following few sections describe how to build up these objects in the DC software.

B.4.1 Listing of FreeParameters

The FreeParameter class describes objects associated with the model parameters.

```
>> fp = FreeParameter(NAME,RANGE);
```

creates a FreeParameter object. `NAME` is a char array denoted the parameter's name (each parameter should have a unique name) and `RANGE` is a 1x2 numeric array denoted the lower and upper limits of the parameters value.

```
>> fp = FreeParameter(NAME,NOMINAL,UNCERTAINTY);
```

creates a FreeParameter object, using a nominal value with uncertainty. `NOMINAL` is a scalar value. `UNCERTAINTY` can be a scalar such that the parameter is in the range `NOMINAL+/-UNCERTAINTY`, or it can be a 1x2 numeric array so that the parameter is in the range `[NOMINAL+UNCERTAINTY(1), NOMINAL+UNCERTAINTY(2)]`. Uncertainty can also be specified relatively or with log transformations. The behavior of these types of uncertainty specification are not as stable, however. If you are interested, read the command line help for the FreeParameter object.

The FreeParameter objects can be stacked into vertical arrays. If you would like to create the FreeParameters using a for-loop, you can pre-allocate empty FreeParameter arrays. For example,

```
>> fps = FreeParameters(5);
```

initializes a 5x1 FreeParameter object array. This can then be filled using one of the above syntaxes. For example,

```
>> fps(1) = FreeParameter('param1',[3 10]);
```

B.4.2 ResponseObservations

The observation of some scalar value of interest (the observable) related to an experimental response is formulated as a ResponseObservation object.

```
>> ro = ResponseObservation(VALUE,UNCERTAINTY);
```

`VALUE` is a numeric scalar representing the measured observation. `UNCERTAINTY` can either be a scalar or a 1x2 double. If it is a scalar, then the assertion is that the observable `y` is constrained as $|$ `y - VALUE` $| \leq$ `UNCERTAINTY`. If `UNCERTAINTY` is a 1x2 double then the assertion is that `UNCERTAINTY(1)` $\leq$ `y - VALUE` $\leq$ `UNCERTAINTY(2)`. `UNCERTAINTY` can also be denoted as relative or with log transformations, but as with the FreeParameter object, we suggest you use the above syntax.

B.4.3 ResponseModels

A response model is an algebraic function or simulation code that maps a vector of fixed parameter values to the scalar observable of the simulation response. For example, if the model simulates some response over time, the scalar observable may be the maximum value, or the rise-time to the peak value. Each model represents a different observable. The model does not necessarily depend on all FreeParameters, but it does depend on a subset of them.

There are two main types of ResponseModel objects that can be created. The first is an algebraic model; specifically, it is an affine or quadratic function of the parameters. The second is a general simulation model. Their creation is outlined in the next two sections.

B.4.3.1 Linear or Quadratic Models

```
>> rm = ResponseModel(COEFFS,MODELDOMAIN);
```

creates an algebraic ResponseModel object.

`MODELDOMAIN` is an Nx1 struct array with two fields, `name` and `range`. N is the number of parameters that this model depends on. For the i^{th} element of `MODELDOMAIN`, `.name` is a char array, that matches the name of one of the FreeParameter objects and `.range` is the range of this parameter over which the model is valid. The range must be a superset of the range define in the corresponding FreeParameter object. The `MODELDOMAIN` may only contain parameters described in the FreeParameter objects. The order does not need to correspond to the order of the FreeParameter array in any way. The order of parameters in `MODELDOMAIN` is the order in which the ResponseModel expects parameters for calculation.

If the ResponseModel is affine, `COEFFS` will be an (N+1)x1 numeric array, such that the model corresponds to the form

```
y = COEFFS'*[1;x];
```

where `x` is a Nx1 vector of parameters in the same order as `MODELDOMAIN`.

If the ResponseModel is quadratic, `COEFFS` will be an (N+1)x(N+1) symmetric numeric array, such that the model corresponds to the form

```
y = [1 x']*COEFFS*[1;x];
```

where `x` is a Nx1 vector of parameters in the same order as `MODELDOMAIN`. This is the same as

```
y = COEFFS(1,1) + 2*COEFFS(1,2:end)*x + x'*COEFFS(2:end,2:end)*x;
```

The ResponseModel can also be specified with output uncertainty/error as

```
>> rm = ResponseModel(COEFFS,MODELDOMAIN,OUTPUTUNC);
```

If the algebraic form is a surrogate fit, then `OUTPUTUNC` may be the fitting error. For example, this would be the peak fitting error over the model domain when the algebraic model is a low-order approximation to a more detailed model. `OUTPUTUNC` should be a scalar if the uncertainty in the model output is symmetric. If it is asymmetric this should be a 1x2 vector, with `OUTPUTUNC(1) <= eta(x) - M(x) <= OUTPUTUNC(2)`, where `M(x)` is the algebraic model and `eta(x)` is the true model.

B.4.3.2 Models Using Simulation m-files

```
rm = ResponseModel(MODELHANDLE);
```

creates a response model given the function handle `MODELHANDLE`. It is also possible to create a ResponseModel object with some additional inputs.

```
rm = ResponseModel(MODELHANDLE,ADDLINPUT1,...,ADDLINPUT2);
```

The function that is pointed to by `MODELHANDLE` has a special form. The easiest way to learn this form is to use the template file "dcModel.m" found in the "fileTemplates" subdirectory of the main DC directory. If `MODELHANDLE` is `@myModel`, then the first line of "myModel.m" is

```
function out = myModel(flag,paramMatrix,varargin)
```

or something similar. This function not only needs to return evaluations of a set of parameter values, but also return information about the function. This first input, `flag`, will be a char array denoting the information requested by the toolbox's methods. The following list summarizes possible inputs for `flag`, and what the corresponding output needs to be.

`'simulate'` (Required) The input `paramMatrix` will be an NxL matrix (N is the number of parameters for the model and L the number of points) where each column represents one parameter vector. The output should be a 1xL array of values.

`'getModelDomain'` (Required) This should return a Nx1 structure with fields `.name` and `.range` corresponding to the parameters used by the model in the order it expects them. See the description above for the case when the ResponseModel is created with an algebraic form.

`'isSavedEnabled'` (Optional, default=false) This should return either true or false. If true, evaluations of this model are saved in the "savedEvaluations" subdirectory of the main DC directory.

`'isMultipleResponsesEnabled'` (Optional, default=false) This should return either true or false. If true, then this single file returns multiple responses (or features) for each simulation. If there are p responses then the output in the `'simulate'` case should have dimension pxL. Additionally, a response list must be defined as in the next flag. We very highly recommend that this feature only be used when `'isSavedEnabled'` is set to true.

`'getResponseList'` (Required if `'isMultipleReponsesEnabled'` is true) This case returns a px1 cell array chars. Each entry corresponds to the one-word name of one of the features. This should be in the same order as returned by the `'simulate'` case.

`'getName'` (Optional, default=filename) If the model can be different depending on the additional inputs then this lets you define a character array (string) that will be used to name files when saving points. Note: if `'isMultipleResponseEnabled'` is true this should NOT reflect a specific feature. Feature names are handled automatically. This name can depend on the other input arguments to this function as they are passed in when getting this string.

There are a few other possible flags, that are optional, and described in the template file. You do not need to provide output for the optional flags, the toolbox will catch any errors when requesting this information, and assume defaults. The additional inputs are passed in to the function each time, so they may be used to determine the model domain structure, or name (for example).

When multiple features is enabled, each feature gets its own ResponseModel object, even though they share an m-file. In this case, the first additional input will be the char array specifying the feature. For example, imagine a model simulated by "car.m" based on various driving and road conditions. It has two features we're interested in, `'top_speed'` and `'mileage'`. Then the two response models would be created via

```
>> rm1 = ResponseModel(@car,'top_speed');

>> rm2 = ResponseModel(@car,'mileage');
```

How the m-file simulates the model is entirely up to you. It can run basic MATLAB code. It can call other files. If you have the means for MATLAB to call another language (e.g. C-code via MATLAB's mex interface), the simulation can be in that other language. You can even use it to parallelize the simulation at the various parameter vectors. So all this m-file is, is a MATLAB wrapper file for whatever simulation code you would like to use.

B.4.4 DCDataset

The last few steps in creating a dataset, is to pair up the ResponseObservation objects with their corresponding ResponseModel objects and putting everything together with the FreeParameters.

First, for each observable, create a ModelAndObservationPair object.

```
>> mop = ModelAndObservationPair(RO,RM,NAME);
```

where `RO` is a ResponseObservation object, `RM` is a ResponseModel object, and `NAME` is a char array. These objects should be vertically concatenated into a ModelAndObservationPair array. If these objects are being created in a for-loop, you can pre-allocate the array using the syntax

```
>> mops = ModelAndObservationPair(m);
```

which will create an mx1 array.

Once an mx1 ModelAndObservationPair array `mops` and a nx1 FreeParameter array `fps` have been created, a DCDataset object can be formed.

```
>> dset = DCDataset(mops,fps);
```

This object is the basis for all further analysis.

To create a dataset with no model/experiment data, simply create an empty ModelAndObservationPair by using the constructor with no inputs.

```
>> dset = DCDataset(ModelAndObservationPair,fps);
```

B.5 Surrogate Model Formulation

During the calculation of the consistency measure and response prediction (see §B.6) and parameter optimization (see §B.7), for each ResponseModel that is not

quadratic (or affine) the DC toolbox generates a quadratic surrogate model. This happens automatically. The subdivision of the prior knowledge parameter domain will be discussed in Section B.8. This section discusses how a quadratic is fit for each ResponseModel on each subdomain.

Surrogate models and all fitting information is stored in a PolyDataset object, which inherits from the DCDataset class.

B.5.1 Active Parameters and Parameter Transformations

Before the surrogate fit is made, some evaluations of the ResponseModel are made to help determine "active" parameters. These are the parameters which are most essential to the change in the ResponseModel's output. Those parameters that are considered not active, are set at their nominal values and then ignored (the surrogate fit will not depend on them). This can reduce fitting time, as less evaluations are needed to make a fit, but will potentially add fitting error.

Also to improve fit accuracy, the quadratic surrogate may have a logarithmic dependence on some of its parameters, meaning the surrogate depends on the $\log_{10}$ of some parameters. Which parameters are chosen to be log can be determined automatically.

B.5.2 Iterative Fitting Algorithm for Surrogate Convergence

On the current subdomain, a quadratic might fit quite poorly. The fit will eventually be improved through a branch and bound algorithm. However, straight assessment of the fitting error will not gage whether this fit is the best possible quadratic fit. The iterative fitting algorithm continues to add samples to the regression, until the quadratic fit stops changing. When the surrogate has "settled," errors are assessed.

Two quadratic forms Q_1 and Q_2 are compared over the domain $\mathcal{H}$ by evaluating the normalized deviation between the functions

$$\text{deviation} = \frac{\int_{\mathcal{H}} Q_1(\mathbf{x}) - Q_2(\mathbf{x})\ d\mathbf{x}}{\int_{\mathcal{H}} Q_2(\mathbf{x})\ d\mathbf{x}}$$

If the deviation has not met a tolerance, then more points are added to the regression to create a new fit.

B.5.3 Evaluation Storage for Rapid Restarts

When the `'isSavedEnabled'` flag of an m-file based ResponseModel is set to true, then evaluations of that model are stored in the "savedEvaluations" subdirectory of the main DC directory. When a single m-file produces multiple responses, then all responses are saved, even if only one of them is the current response being fitted.

Suppose that an m-file produces multiple responses and evaluations are flagged for saving. When the surrogate model is generated for the ResponseModel associated with one of the responses, all responses are saved to the hard disk. Subsequently, when the ResponseModel for another response is generated, the points are loaded from the disk, thereby reducing, and possibly eliminating the need for further evaluation of that m-file.

If the computation is restarted, any saved evaluations can be reused for fitting. Thus reducing the computation for creating surrogate fits from scratch. This also provides a pseudo-backup for power outages and system crashes that occur during long computations.

WARNING: If at any point, the simulation files are changed, you need to make sure to clear all saved files. This can be done by either deleting the files in the "savedEvaluations" subdirectory (and NOT the "savedEvaluationsDir.m") or by using a method:

```
>> deleteSavedEvaluations(dset);  %clears evaluations
                                  %for a DCDataset
>> deleteSavedEvaluations(rm);    %clears evaluations
                                  %for a single ResponseModel
```

B.5.4 Fitting Error Approximation

Once a quadratic surrogate has been generated, the maximum fitting error over the domain is estimated. This is done with a combination of determining the error of the fit points and validation points and via local searches with `fmincon`. The maximum error found from these three methods is then added to the ResponseObservation error to get the total error for a ModelAndObservationPair.

B.6 Response Prediction and Consistency Measure Calculations

Once a DCDataset has been created a consistency analysis can be performed by creating ConsistencyTest object.

```
>> ctestObj = ConsistencyTest(dset);
```

This calculates the **relative** consistency measure, $\mathrm{C}_{\mathcal{D}}$

$$\mathrm{C}_{\mathcal{D}} := \max\ \gamma$$
$$\text{s.t.} \begin{cases} l_e(1-\gamma) + l_{e,\text{fit}} \leq M_e(\mathbf{x}) - d_e \leq u_e(1-\gamma) + u_{e,\text{fit}},\ e = 1, \ldots, m \\ \mathbf{x} \in \mathcal{H} \end{cases}$$

where l_e, $l_{e,\text{fit}}$, u_e, and $u_{e,\text{fit}}$ are the lower and upper experiment and surrogate fitting errors, d_e is the ModelObservation value, M_e is a ResponseModel, and $\mathcal{H}$ is the bounds defined by the FreeParameter object. We stress that this is the relative consistency measure, because it was defined in an absolute sense in some of our published work. If the consistency measure is positive, the dataset is said to be consistent, if the consistency is negative it is inconsistent. Further, the consistency measure is the percentage that all experiment uncertainty must be reduced in order to achieve consistency.

Likewise, if a DCDataset is constructed, we can make predictions of the range of values that an additional ResponseModel can take over the set of constrained feasible parameters. This is posed as the creation of a ResponsePrediction object.

```
>> predObj = ResponsePrediction(rm0,dset);
```

Creating this object solves two optimization problems

$$L_0 := \min\ M_0(\mathbf{x})$$
$$\text{s.t.} \begin{cases} l_e + l_{e,\text{fit}} \leq M_e(\mathbf{x}) - d_e \leq u_e + u_{e,\text{fit}},\ e = 1, \ldots, m \\ \mathbf{x} \in \mathcal{H} \end{cases}$$

$$R_0 := \max\ M_0(\mathbf{x})$$
$$\text{s.t.} \begin{cases} l_e + l_{e,\text{fit}} \leq M_e(\mathbf{x}) - d_e \leq u_e + u_{e,\text{fit}},\ e = 1, \ldots, m \\ \mathbf{x} \in \mathcal{H} \end{cases}$$

where M_0 is the function corresponding to the predicted ResponseModel.

Both the consistency and predictions can be customized using a DCOptions object (see §B.10).

```
>> ctestObj = ConsistencyTest(dset,opts);

>> predObj = ResponsePrediction(rm0,dset,opts);
```

In each calculation, quadratic surrogate models are fitted where required, and the problem is formulated as a nonconvex quadratically constrained quadratic program (NQCQP). Techniques for solving the NQCQP provide both outer and inner bounds on the solutions.

B.6.1 NQCQP Formulation & Bounding

After surrogate fitting, the problems are formulated as Nonconvex Quadratically Constrained Quadratic Programs. Outer bounds (upper bound on a maximization and a lower bound on a minimization) are solved for using what is known as the S-procedure. This formulates the problem as a semidefinite program which is solved with the SeDuMi optimization package.

Inner bounds (lower bound on a maximization and an upper bound on a minimization) are solved using a call to `fmincon`. Once an optimal parameter vector is found for the quadratic surrogates, the original model is evaluated to create a true inner bound.

B.6.2 Certification of Results

Once computation of the ConsistencyTest object or the ResponsePrediction object is complete, the objects contain a large amount of information regarding the calculation. This includes, but is not limited to,

- All branching locations for the branch and bound algorithm (see §B.8)
- All surrogate fits at each iteration of the branch and bound algorithm, with fitting errors, and number of function evaluations
- Bounds on the objective for each subdomain during the branching
- Sensitivities of the optimal value to experiment and parameter uncertainty (see §B.6.2.1).

The `report` method will generate a report on the calculations in HTML format (which can be viewed with a web-browser) in the current directory.

```
>> report(ctestObj);

>> report(predObj);
```

The bounds on the consistency measure are retrieved via

```
>> LB = ctestObj.LB; %lower bound on measure
>> UB = ctestObj.UB; %upper bound on measure
```

The bounds on the response prediction interval are retrieved in a similar fashion.

```
>> LBo = predObj.LBo; %outer bound on minimum
>> LBi = predObj.LBi; %inner bound on minimum
>> UBi = predObj.UBi; %inner bound on maximum
>> UBo = predObj.UBo; %outer bound on maximum
```

Furthermore, the parameter vectors that give the inner bounds are available.

```
>> LBx = predObj.LBx;
>> UBx = predObj.UBx;
```

B.6.2.1 Sensitivities

Outer bound computations automatically produce Lagrange multipliers. These are extracted from the object by

```
>> ctestMults = ctestObj.upperBndMults;
>> predMults = predObj.outerBndMults;
```

The prediction multipliers are in a structure with two fields `.lower` for the minimization problem and `.upper` for the maximization problem. `ctestMults`, `predMults.lower`, and `predMults.upper` are each a struct object with four fields: `.paraml`, `.paramu`, `.expl`, and `.expu` representing the multipliers for the lower and upper bound constraints for the parameter and experiments.

The objects can also convert the Lagrange multipliers to be sensitivities. These are the partial derivatives of the optimal outer bound objective with respect to the uncertainty bounds.

```
>> ctestSens = ctestObj.upperBndSens;
>> predSens = predObj.outerBndSens;
```

These are struct objects with the same form as the multiplier structs.

B.7 Parameter Optimization

Parameter optimization is a process of selecting a vector of parameters that minimizes some objective. The DC toolbox has a parameter optimization algorithm using the NQCQP techniques.

```
>> paramOptimObj = ParameterOptimization(dset);
```

Attempts to find the vector of parameter values $\mathbf{x}$ that solves the problem

$$\min_{\mathbf{x}\in\mathcal{H}} \sum_{e=1}^{m} w_e(M_e(\mathbf{x}) - d_e)^2$$

where w_e are weights that are by default set to $1/u_e$. This does so in the same general method as consistency and prediction. It creates polynomial fits where required, constructs an NQCQP, and computes upper and lower bounds on the objective. These bounds are retrieved via

```
>> LB = paramOptimObj.costLB;
>> UB = paramOptimObj.costUB;
```

The optimal parameter vector is retrieved via

```
>> x = paramOptimObj.bestx;
```

A DCOptions can be provided to set various options.

```
>> paramOptimObj = ParameterOptimization(dset,opts);
```

It is also possible to minimize the 1-norm or the infinity-norm instead of the 2-norm. This is done with

```
>> paramOptimObj = ParameterOptimization(dset,opts,norm);
```

where `norm` is either `'one'`, `'two'`, or `'inf'`.

Furthermore, you can specify the weights w_e as a mx1 vector (m being the number of ModelAndObservationPairs in the DCDataset).

```
>> paramOptimObj = ParameterOptimization(dset,opts,norm,weights);
```

B.8 Automatic Branch and Bound Algorithm

A branch and bound algorithm is provided for the computation of the ResponsePrediction, ConsistencyTest, and ParameterOptimization objects. By default, the computations do not perform any branching, but by setting `'maxBranchBoundIter'` option in the DCOptions object to 2 or greater (see §B.10), the branch and bound algorithm will performed automatically. The goal is to improve the surrogate fits by examining smaller subdomains (making the surrogate a piecewise quadratic) and to reduce the gap between inner and outer bounds.

Even when quadratic surrogates are already created, the branch and bound algorithm can improve the gap between inner and bounds that is due to the convex relation made the outer bounds.

B.8.1 Piecewise Surrogate Model Binary Tree

The branch and bound algorithm divides the domain into subdomains, and keeps track of each of the relevant subproblems in a binary tree data structure. This structure is contained by the PiecewiseSurrogateModelTree object. The user generally will not see, nor deal with this object, but it is available in the ResponsePrediction, and ConsistencyTest objects.

Each node of the tree represents a subdomain of the prior knowledge parameter domain. The children of a node represent the partition of the node into subdomains. Each division is along a single parameter. Each node includes surrogate fits and optimization bounds. For a minimization, the optimal value is the minimum of the optimal values over each of the leaves of the tree.

The tree structure provides a history of the algorithms behavior. It tracks where splits are made, the intermediate surrogate fits, and the optimal values.

B.9 Warm Starting Using Previously Calculated Piecewise Surrogates

Suppose that you run a consistency analysis, and results show that the consistency measure is bounded in the interval $[-0.23,\ 0.3]$. More iterations of the branch and bound algorithm should be able to resolve this gap. However, we have potentially spent lots of time generating piecewise surrogate models for each of the ResponseModels in the DCDataset. However, we do not have to completely restart the algorithm. The ConsistencyTest constructor can take a ConsistencyTest object as one of its inputs to warm start the branch and bound iteration.

```
>> ctestObj = ConsistencyTest(dset);

>> opt = DCOptions('maxBranchBoundIter',2);

>> ctestObjNew = ConsistencyTest(ctestObj,opt);
```

This can also be done with the ResponsePrediction objects and ParameterOptimization objects. Furthermore, the ResponsePrediction constructor can be warm started with a ConsistencyTest object.

```
>> ctestObj = ConsistencyTest(dset); %first verify consistency

>> predObj = ResponsePrediction(rm0,ctestObj);
```

B.10 Fine Tuning Via User Options

There are many options that a user can set when creating a ConsistencyTest, ResponsePrediction, or ParameterOptimization object. These are set using the DCOptions object.

```
>> opts = DCOptions;
```

This creates a DCOptions object with all options set to their default. If no DCOptions object is provided to the algorithm, this is the default that is generated internally.

Options are set in one of two ways. The first is to list them during creation as property/value pairs.

```
>> opts = DCOptions(PROPERTY1,VALUE1,PROPERTY2,VALUE2,...)
```

The second way is to use dot-referencing to change the defaults or current options in an already created object. For example, say we wanted to turn off the `'display'` option.

```
>> opts = DCOptions;

>> opts.display = 'none';
```

The following is a list of all options the user can set, including their possible values and meaning. First a simple option for the display level.

'display' (default='iter'). Can be 'off', 'final', 'notify', 'iter', 'all', or 'ALL'. Defines the amount of information printed to the screen during the algorithms.

The following properties affect the optimization algorithms.

'omitInnerBound' (default=false) Can be true or false. If true, inner bounds are not calculated. However, if 'maxBranchBoundIter'>1, then inner bounds are calculated once for the branch and bound algorithm.

'omitOuterBound' (default=false) Can be true or false. If true, outer bounds are not calculated.

'maxBranchBoundIter' (default=1) Can be any positive integer. Maximum number of iterations of the branch and bound algorithm. If reached before tolerances are met, the algorithm exits. The first iteration equates to the first calculation before any branching. The second iteration is the calculations after the first branch.

'branchBoundTermTol' (default=0.02) Can be any positive number. The branch and bound algorithm will stop if the gap between inner and outer bounds is smaller than this number.

'nRestart' (default=2) Can be any positive integer. The number of times the inner bound optimizations are restarted with a new seed. The result returned is the optimum over all restarts.

'tolFun' (default=1e-5) Can be any positive number. This is the function tolerance used by the inner bound calculations (passed to fmincon).

'tolCon' (default=1e-5) Can be any positive number. This is the constraint tolerance used by the inner bound calculations (passed to fmincon).

'sedumiParEps' (default=1e-9) Can be any positive number. This is the optimization tolerance used by outer bound calculations (passed to SeDuMi).

'constraints' (default=[1 0 0 0]) A 1x4 array of 1's and 0's. Defines which transformations to perform the optimization on. The first column is linXlinY, the second is logXlinY, the third is linXlogY, and the fourth column is logXlogY, where X is the parameters and Y is the model outputs. 1 means include the indicated transformation in the optimization, 0 means exclude them. See the 'surfaceTransformation' option.

The following properties affect how surrogate fits are made.

'fitNorm' (default=2) Can be 2 or inf (the values, not the strings). This is the norm of residual fitting error that is to be minimized by the fit. Ultimately, the infinity-norm error is what is added to experiment uncertainties in the optimization. However, the infinity-norm problem takes longer to solve, and for large problems can run into memory issues.

'findIOTransforms' (default=true) Can be true or false. If true, uses some function evaluations to determine which parameters should have a $\log_{10}$ transformation for the best surrogate fit. Similarly, determines if the output should have a $\log_{10}$ transformation.

'activeParamSelCutOff' (default=0.05) Can be any positive number. A tolerance used for truncating the active parameter list. Setting this value to zero will insure all model parameters are used in the corresponding surrogate model. 100*activeParamSelCutOff roughly corresponds to the percent error introduced by eliminating nonactive parameters. We suggest never exceeding 0.2.

'nPntsPerCoeff4ActiveParamSel' (default=25) Can be any positive integer. Corresponds to the number of function evaluations per coefficients in the surrogate fit that are computed to determine active parameters.

'surfaceFittingMode' (default='iterative') Can be 'iterative' or 'oneShot'. If 'iterative', then surrogates are fitted using the iterative fitting algorithm (see §B.5.2). If 'oneShot', then surrogates are fit only once per iteration of the branch and bound algorithm.

'plotFitProgress' (default='off') Can be 'off' or 'on'. If on, and 'surfaceFittingMode' is set to 'iterative', then the difference metric between successive surrogate fits during the iterative fitting algorithm is plotted.

'minFitIter' (default=3) Can be any positive integer less than or equal to the 'maxFitIter' option. The minimum number of iterations of the iterative fitting algorithm to perform if 'surfaceFittingMode' is set to 'iterative'.

'maxFitIter' (default=7) Can be any positive integer greater than or equal to the 'minFitIter' option. The maximum number of iterations of the iterative fitting algorithm to perform if 'surfaceFittingMode' is set to 'iterative'.

'nSuccessfulFitIter' (default=2) Can be any positive integer. The number of times successive iterations a fit must meet the 'fitConvergenceTol' on the error convergence, before it is considered a good fit (if 'surfaceFittingMode' is set to 'iterative').

'fitConvergenceTol' (default=0.05) Can be any positive number. The tolerance that the metric between successive surrogate fits must meet to be considered "successful" (if 'surfaceFittingMode' is set to 'iterative').

'maxPnts4Fit' (default=Inf) Can be any positive integer. The maximum number of points that can be used for a surrogate fit.

'nPntsPerCoeff4OneShot' (default=20) Can be any positive integer. When 'surfaceFittingMode' is set to 'oneShot', this option determines the number of evaluations to use per coefficients in the surrogate to create the surrogate fit.

`'subspaceDiscovery'` (default=`'off'`) Can be `'off'` or `'on'`. When `'on'` and when `'surfaceFittingMode'` is set to `'oneShot'`, the surrogate fitting algorithm attempts to find a lower dimensional active subspace upon which the model depends.

`'subspaceThreshold'` (default=0.1) Can be any positive number. When `'subspaceDiscovery'` is set to `'on'`, this option determines the cutoff of the singular values of a gradient matrix of the ResponseModel, thus determining the dimension of the active subspace. The cutoff is the first singular value σ_i that satisfies `opt.subspaceThreshold`$*\sigma_1 > \sigma_i$.

`'derivRange'` (default=0.01) Can be any positive number. When `'subspaceDiscovery'` is set to `'on'`, this option determines the spread of evaluations used to approximate gradients of the ResponseModel. If the parameter domain is normalized to be $[-1,\ 1]^n$, then `opt.derivRange` is the radius of the circle in these coordinates that contains all points used to estimate a gradient. Larger values may be required for ResponseModels with lots of noise.

`'surfaceTransformation'` (default=`{'linXlinY'}`) This is a cell array of strings listing the different transformations to be used when making a fit. This must be a 'superset' of the `'constraints'` option. Valid values are any cell array containing one or more of the following: `'linXlinY'`, `'logXlinY'`, `'linXlogY'`, and `'logXlogY'`.

`'useAllPnts4Fit'` (default=true) Can be true or false. After loading saved evaluations, the fitting algorithm will use all points already available to create the fit if this option is set to true. If set to false, the algorithm will only use the amount it would have computed had there been no saved points.

`'nLocalValidationSearches'` (default=3) Can be any nonnegative integer. When validating a surrogate fit, the algorithm uses `fmincon` to search for the location of the worst error. This option determines the number of restarts of this procedure.

`'nPntsPerCoeff4Validation'` (default=250) Can be any nonnegative integer. When validating a surrogate fit, the algorithm evaluates the ResponseModel and the surrogate at a set of validation points to look for average and worst case errors. This option determines the number of evaluations per coefficient in the surrogate for this validation.

B.11 Examples

The toolbox comes with several built-in examples. These are very useful for demonstration of the setup and basic operations associated with the toolbox. They

can be found in the "examples" subdirectory of the main DC toolbox directory. The following is a list and brief description of each example. We recommend that you look at the m-file code for each example to see what it is doing and to read the comments before running the example. It may also be useful to run the examples one cell at a time using MATLAB's "code cell" features.

Each example requires that the proper directories are added to the path via the `DCsetup` script.

General Demos: In the "generalDemos" subdirectory of the "examples" directory, there are 3 examples. The three demo files — "demo1_linearModels.m", "demo2_quadraticModels.m", and "demo3_generalModel.m" — demonstrate creating some toy DCDatasets using each of the three types of ResponseModel, linear, quadratic, and general. These demos do nothing more than create the DCDataset object.

GRI-Mech 3.0: In the "GRI" subdirectory of the "examples" directory is a demo relating the GRI-Mech 3.0 dataset. This is a dataset of 77 experimental observables, with data and pre-created quadratic surrogate models. There are 102 parameters in the system. The file "gri_mech_demo.m" includes a demonstration of consistency testing, sensitivity analysis, and prediction with this system. Since all model are already quadratic, the analysis is relatively quick; however, it still takes a few minutes due to the large size of the system.

Simple nonlinear dynamic system: In the "Prajna" subdirectory of the "examples" directory is a modified example presented by Stephen Prajna at the 2003 IEEE Conference on Decision and Control. This example has a nice HTML file in the "html" subdirectory of the "Prajna" directory called "runPrajnaExample.html" that can be viewed in a web browser. The HTML file shows the output and comments from the file "runPrajnaExample.m." This example demonstrates a consistency check on the dataset, and uses multiple features for nonquadratic models as well as evaluation saving.

Mass-Spring-Damper System: In the "massSpringDamper" subdirectory of the "examples" directory is an example involving the displacement of a mass attached to a spring and damper when a step input force is applied. The example does not have quadratic models. There are 3 features examined, and the model evaluations are saved on the hard disk. We suggest looking not only at the example m-file "runMSDexample.m," but also the model file "msdLinearModel.m."

B.12 Conversion From Version 1.0

Conversion from version 1.0 to version 2.0 is pretty simple, albeit, possibly tedious. The main thing you need to worry about is to make sure to run DCsetup at the

beginning of each MATLAB session, and to make sure that if any DC objects are created inside a function to include the command `import DClab.*`. Lastly, Table B.1 shows the object equivalence from version 1.0 to version 2.0.

Table B.1: Object conversion table, showing the names of equivalent objects from versions 1.0 and 2.0 of the Data Collaboration toolbox.

Version 1.0	Version 2.0
ParameterAssertion	FreeParameter
ExperimentAssertion	ResponseObservation
ModelAssertion	ResponseModel
DatasetUnit	ModelAndObservationPair
ConsistTest	ConsistencyTest
Prediction	ResponsePrediction
ParameterOptimization	ParameterOptimization
DCOptions	DCOptions

All other objects are internal, and the user will most likely not interact with directly.

B.13 Troubleshooting, Help, and Feature requests

There are several common mistakes that are made when using the Data Collaboration toolbox that can lead to errors or incorrect results. They include

- If your models are flagged for saving evaluations, make sure that all old evaluations are cleared when you make changes to the model file.
- Check the units on the parameters in your models and in your FreeParameter objects. The analysis assumes that the FreeParameters are defined in the same units as the ResponseModels (including transformations). The analysis also assumes the output of the models is in the same units as the experiment data and uncertainty with the default behavior. It is possible for the data and uncertainty to defined in terms of the log of the ResponseModel output (see the ResponseObservation command line help).
- Each FreeParameter should have a unique name, that is case-dependent.
- The FreeParameter array that is given to the DCDataset constructor should include all FreeParameters listed by the `MODELDOMAIN` structures of the ResponseModels.

Each of the objects and their methods have command line help. This can be accessed by typing `help` followed by a space and the function name. For example to read about the DCOptions object and all its properties, type

```
>> help DCOptions
```

If you have any more questions, come across any bugs, have suggestions, or want to suggest features, please visit http://collab-sci.sourceforge.net/ and click on the "Tracker" link.