

Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-21-2008

Improving Mixed Variable Optimization of Computational and Model Parameters Using Multiple Surrogate Functions

David Bethea

Follow this and additional works at: <https://scholar.afit.edu/etd>



Part of the [Design of Experiments and Sample Surveys Commons](#), and the [Operational Research Commons](#)

Recommended Citation

Bethea, David, "Improving Mixed Variable Optimization of Computational and Model Parameters Using Multiple Surrogate Functions" (2008). *Theses and Dissertations*. 2664.
<https://scholar.afit.edu/etd/2664>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.



**IMPROVING MIXED VARIABLE OPTIMIZATION OF
COMPUTATIONAL AND MODEL PARAMETERS USING
MULTIPLE SURROGATE FUNCTIONS**

THESIS

David Bethea, Captain, USAF

AFIT/GOR/ENC/08-01

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY**

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT/GOR/ENC/08-01

**IMPROVING MIXED VARIABLE OPTIMIZATION OF
COMPUTATIONAL AND MODEL PARAMETERS USING
MULTIPLE SURROGATE FUNCTIONS**

THESIS
David Bethea
Captain, USAF

AFIT/GOR/ENC/08-01

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense or the United States Government.

AFIT/GOR/ENC/08-01

**IMPROVING MIXED VARIABLE OPTIMIZATION OF
COMPUTATIONAL AND MODEL PARAMETERS
USING MULTIPLE SURROGATE FUNCTIONS**

THESIS

Presented to the Faculty
Department of Operational Sciences
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Operations Research

David Bethea, B.S.

Captain, USAF

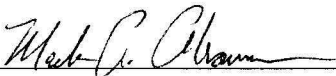

March 2008

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**IMPROVING MIXED VARIABLE OPTIMIZATION OF
COMPUTATIONAL AND MODEL PARAMETERS
USING MULTIPLE SURROGATE FUNCTIONS**

**David Bethea, B.S.
Captain, USAF**

Approved:

	<u>20 MARCH 2008</u>
Lt. Col. Mark A. Abramson, Ph.D. Chairman	Date
	<u>21 MAR 08</u>
James W. Chrissis, Ph.D. Member	Date

Abstract

This research focuses on reducing computational time in parameter optimization by using multiple surrogates and subprocess CPU times without compromising the quality of the results. This is motivated by applications that have objective functions with expensive computational times at high fidelity solutions. Applying, matching, and tuning optimization techniques at an algorithm level can reduce the time spent on unprofitable computations for parameter optimization. The objective is to recover known parameters of a flow property reference image by comparing to a template image that comes from a computational fluid dynamics simulation, followed by a numerical image registration and comparison process. Mixed variable pattern search and mesh adaptive direct search methods were applied using surrogate functions in the search step to produce solutions within a tolerance level of experimental observations. The surrogate functions are based on previous function values and computational times of those values. The use of multiple surrogates at each search step provides parameter selections that lead to improved solutions of an objective function evaluation with less computational time. Previously computed values for the objective function and computation time were used to compute a time cut-off parameter that allows termination during an objective function evaluation if the computational time exceeded a threshold or a divergent template image was created. This approach was tested using DACE and radial basis function surrogates within the NOMADm MATLAB[®] software. The numerical results are presented.

Dedication

I dedicate this thesis to my wife and son. Their patience and constant reinforcement did not go unnoticed. I also want to thank my family and friends for their support throughout my life and all of my endeavors.

Acknowledgements

I would like to thank the members of my thesis committee, Lt. Col. Abramson and Dr. James W. Chrissis for their efforts in helping accomplish this thesis. I must express my gratitude to my professor Lt. Col. Abramson for his support, patience, and understanding while supervising me. His vast knowledge and expertise in many areas provided an extraordinary technical basis to be taught from. The exceptional mentorship and generosity of his time was greatly valued. I would also like to thank those at Los Alamos National Laboratory, Dr. Thomas J. Asaki and Dr. Matthew J. Sottile who continuously provided test cases and supported this thesis. Their constant assistance was highly appreciated. A special thanks is given to Dr. John E. Dennis Jr. for his guidance and expert advice during my thesis. It was a true honor.

I would also like to give thanks to my peers. Capt Bryan Sparkman was always there when I was there, working very late nights and always helping. Capt Ryan Kappedal was also always there offering words of wisdom and ways to reduce stress. Capt Robert Johnson was always available for the many statistical questions and he was constantly willing to help with different areas throughout this research. Capt Ryan Ponack was a true friend that always found a way to get me to do the things I enjoy outside of work. A special thanks to Capt Todd Paciencia for his patience and helpfulness with DACE and RBF surrogates, among the many other areas he helped me throughout my thesis.

David Bethea

Table of Contents

	Page
Abstract	iv
Acknowledgements	vi
List of Figures	x
List of Tables	xii
1. Introduction	1-1
1.1 Background and Motivation	1-2
1.2 Parameter Optimization Framework	1-4
1.2.1 Computational Fluid Dynamics Simulation	1-6
1.2.2 Numerical Image Registration	1-10
1.2.3 Dual Process Objective Function	1-14
1.3 Purpose	1-16
1.4 Overview	1-16
2. Relevant Literature	2-1
2.1 Optimization Using Surrogate Functions	2-1
2.1.1 Simplified Physics Models	2-2
2.1.2 Functional Data Fit Models: Design and Analysis of Computer Experiments (DACE)	2-4
2.1.3 Functional Data Fit Models: Radial Basis Function (RBF) Surrogates	2-13
2.1.4 Surrogate Management Framework	2-17
2.2 Generalized Pattern Search	2-18
2.2.1 Mixed Variable Pattern Search	2-23

	Page	
2.3	Mesh Adaptive Directed Search	2-27
2.3.1	Mixed Variable Mesh Adaptive Direct Search	2-31
2.4	Surrogates Based on CPU Time	2-32
2.5	Conclusion	2-35
3.	Methodology	3-1
3.1	Mixed Variable Optimization Problem and Notation	3-1
3.2	Customized Search: Optimization using Multiple Surrogates	3-3
3.3	Composition of Surrogate Functions	3-4
3.3.1	Initial Points	3-5
3.3.2	Regression Model	3-8
3.3.3	Correlation Model and Basis Function	3-9
3.4	Trust Region	3-10
3.5	Implementation	3-12
4.	Implementation and Results	4-1
4.1	Coding and Processing	4-1
4.2	Test Problem 1: Lid-Driven Cavity	4-2
4.2.1	Lid-Driven Cavity Results–MVPS	4-3
4.2.2	Lid-Driven Cavity Results–MVMADS	4-7
4.3	Test Problem 2: Barrier Flow	4-10
4.3.1	Barrier Flow Results–MVPS	4-11
4.3.2	Barrier Flow Results–MVMADS	4-14
4.4	Test Problem 3: Liquid Drop	4-17
4.4.1	Liquid Drop Results–MVPS	4-18
4.4.2	Liquid Drop Results–MVMADS	4-21
4.5	Summary	4-24

	Page
5. Conclusions and Recommendations	5-1
5.1 Summary and Conclusions	5-1
5.2 Future Areas of Research	5-4
Bibliography	BIB-1
Appendix A. Additional Results and Algorithms	A-1
Appendix B. Code for Optimization Framework	B-1
Vita	VITA-1

List of Figures

Figure		Page
1.1.	Parameter Optimization Control Diagram	1-5
1.2.	Parameter Space Search	1-6
1.3.	Computational Fluid Dynamics Simulation Algorithm	1-10
1.4.	Computational Fluid Dynamics Simulation Flow Property	1-10
1.5.	Numerical Image Registration of Simulated Flow Properties	1-11
1.6.	Numerical Image Registration Algorithm	1-14
2.1.	Simplified Physics Surrogate	2-3
2.2.	Response Surfaces of DACE Surrogates	2-13
2.3.	Response Surface of an RBF Surrogate	2-16
2.4.	Surrogate Management Framework Algorithm	2-18
2.5.	Basic GPS Algorithm	2-22
2.6.	Basic GPS Algorithm-Pictorial Representation	2-22
2.7.	Mixed Variable GPS Algorithm	2-26
2.8.	Mixed Variable GPS Algorithm-POLL Step Illustration	2-27
2.9.	Basic MADS Algorithm	2-30
2.10.	A GPS FRAME	2-30
2.11.	A MADS FRAME	2-31
2.12.	Mixed Variable MADS Algorithm	2-32
3.1.	Latin Hypercube Samplings	3-6
3.2.	Inscribed Central Composite Design (2-D)	3-7
3.3.	Trust Region Approach Applied to Surrogates	3-12
3.4.	TA-MVMADS Algorithm	3-13
3.5.	TA-MVMADS Algorithm Continued	3-14

Figure		Page
4.1.	MVPS Lid-Driven Cavity Image Results	4-5
4.2.	MVPS Lid-Driven Cavity Time vs Obj. Function	4-7
4.3.	CFD Simulation Flow Properties of Barrier Flow	4-11
4.4.	MVPS Barrier Flow Image Results	4-12
4.5.	MVPS Barrier Flow Time vs Obj. Function	4-14
4.6.	Lid-Driven Cavity and Barrier Flow Problem Mappings . . .	4-14
4.7.	CFD Simulation Flow Properties of Liquid Drop	4-17
4.8.	MVGPS Liquid Drop Image Results	4-19
4.9.	MVGPS Liquid Drop Time vs Obj. Function	4-21
4.10.	Liquid Drop Problem Mapping	4-21
4.11.	MVMADS Liquid Drop Image Results	4-23
4.12.	MVMADS Liquid Drop Time vs Obj. Function	4-24
A.1.	CFD Simulation Flow Properties of Lid-Driven Cavity	A-2
A.2.	MVGPS Lid-Driven Cavity Obj. and Time Based Surrogates	A-2
A.3.	MVGPS Lid-Driven Cavity Performance History	A-3
A.4.	MVMADS Lid-Driven Cavity Time vs Obj. Function	A-3
A.5.	MVMADS Lid-Driven Cavity Obj. and Time Based Surrogates	A-4
A.6.	MVMADS Lid-Driven Cavity Performance History	A-4
A.7.	MVGPS Barrier Flow Obj. and Time Based Surrogates . . .	A-5
A.8.	MVGPS Barrier Flow Performance History	A-5
A.9.	MVMADS Barrier Flow Time vs Obj. Function	A-6
A.10.	MVMADS Barrier Flow Obj. and Time Based Surrogates . .	A-6
A.11.	MVMADS Barrier Flow Performance History	A-7
A.12.	MVGPS Liquid Drop Obj. and Time Based Surrogates . . .	A-7
A.13.	MVGPS Liquid Drop Performance History	A-8
A.14.	MVMADS Liquid Drop Obj. and Time Based Surrogates . .	A-8
A.15.	MVMADS Liquid Drop Performance History	A-9

List of Tables

Table		Page
4.1.	MVPS: Lid-Driven Cavity Results	4-4
4.2.	MVPS: Lid-Driven Cavity Surrogate Performance	4-6
4.3.	MVPS: Lid-Driven Cavity Iteration Performance	4-6
4.4.	MVMADS: Lid-Driven Cavity Results	4-8
4.5.	MVMADS: Lid-Driven Cavity Surrogate Performance	4-9
4.6.	MVMADS: Lid-Driven Cavity Iteration Performance	4-9
4.7.	MVPS: Barrier Flow Results	4-11
4.8.	MVPS: Barrier Flow Surrogate Performance	4-13
4.9.	MVPS: Barrier Flow Iteration Performance	4-13
4.10.	MVMADS: Barrier Flow Results	4-15
4.11.	MVMADS: Barrier Flow Surrogate Performance	4-15
4.12.	MVMADS: Barrier Flow Iteration Performance	4-16
4.13.	MVPS: Liquid Drop Results	4-18
4.14.	MVPS: Liquid Drop Surrogate Performance	4-20
4.15.	MVPS: Liquid Drop Iteration Performance	4-20
4.16.	MVMADS: Liquid Drop Results	4-22
4.17.	MVMADS: Liquid Drop Surrogate Performance	4-22
4.18.	MVMADS: Liquid Drop Iteration Performance	4-23

IMPROVING MIXED VARIABLE OPTIMIZATION OF COMPUTATIONAL AND MODEL PARAMETERS USING MULTIPLE SURROGATE FUNCTIONS

1. Introduction

In optimization the goal is to find a set of parameters that minimizes or maximizes some objective function f that is typically associated with an application. The optimization problem considered in this research can be expressed as

$$\min_{x \in \Omega} f(x), \quad (1.1)$$

where $f : \Omega \rightarrow (\mathbb{R} \cup \{+\infty\})$ is computationally expensive to evaluate, and the domain Ω is partitioned into continuous and discrete variable spaces Ω^c and Ω^d , respectively. The space of continuous variables is defined by a finite set of linear inequality constraints; namely, $\Omega^c = \{x \in \mathbb{R}^{n^c} : l \leq Ax \leq u, l < u\}$, where $l, u \in (\mathbb{R}^{n^c} \cup \{\pm\infty\})$, $A \in \mathbb{R}^{m \times n^c}$, and n^c is the dimension of Ω^c . The space of discrete variables $\Omega^d \subseteq \mathbb{Z}^{n^d}$ can be represented as a subset of the space of n^d -dimensional integer vectors, where n^d is the dimension. A solution to (1.1) will be denoted by $x^* = (x^{*c}, x^{*d}) \in \Omega$ where $x^{*c} \in \mathbb{R}^{n^c}$ and $x^{*d} \in \mathbb{Z}^{n^d}$, and the optimal solution will be denoted by $f^* = f(x^*)$.

The objective f is treated as a *black box* function, since an analytic expression for f may not be available. Derivatives are also typically unavailable. The function f may also be nonsmooth, discontinuous, and possibly fail to return a value for $x \in \Omega$ [6, 14, 24]. This work is an extension and generalization of the thesis of Magallanez [60], who studied problems in which the computational time required to compute an objective function value $f(x)$ becomes less expensive as x approaches x^* .

1.1 Background and Motivation

Parameter optimization is straightforward in concept in that the parameter space is searched until a result is found that meets a user-defined tolerance. However, this can be effectively impossible in practice, due to the computational cost. For difficult parameter optimization problems that consists of both computational and model variables, it is possible to use mixed variable optimization methods that result in reduced computational costs and improved solutions. In order to study these methods, the class of mixed variable problems targeted here is motivated by an application in which a single objective function evaluation requires two processes: a computational fluid dynamics (CFD) simulation, typically having both computational and model parameters, and a numerical image registration process for comparing CFD simulation output data [60].

In the application, the CFD computational parameters control the fidelity of the temporal and spatial discretizations used in approximating the Navier-Stokes equations for describing fluid flow in a region. Model parameters describe the physical properties of the fluid or the actual simulation. At high temporal and spatial fidelities, computational fluid dynamics simulations exhibit long runtime behavior, thus limiting parameter searches to some type of space-filling sampling usually in a reduced dimension parameter space [16]. Significant computational time is often spent finding intermediate solutions that have little influence on the final result. This misspent time can occur due to a choice of computational parameters that force the simulation to have unnecessarily high fidelity or poor model parameters that inaccurately describe the physical properties of the fluid flow region.

The thesis of Magallanez [60] focused on optimizing only the model parameters of experimental observations through the use of the two aforementioned processes in the objective function, while holding the computational parameters or simulation fidelity variables constant. Keeping the computational parameters constant implies a direct correlation between the objective function value and the computational time

required by the numerical image registration process. The CFD simulation outputs data as an image of a density map or velocity field based on a set of input model parameters. This image is then compared to the experimental data image using the numerical image registration process. Computing resources were carefully monitored through a metric comparison of the computational time of the image registration process.

The motivation of this research is to generalize [60] by extending the parameter optimization to both model and computational parameters. Fine-scale structures often cannot be validated against experimental observables; therefore, the choice of the computational parameters should not be extended any further than necessary for the simulation to reproduce experimental observations [16]. By including computational parameters in the optimization process, the computing resources can be monitored more closely using a metric comparison of the computational times. This also implies that computational time will decrease in direct correlation with the objective function to a certain extent.

At some point during the optimization process, the fidelity of the computational parameters may need to be increased in order to find the optimal solution. This implies a possible increase in simulation time and an inverse correlation; however, the increase in time should ideally be minimal. The minimal simulation fidelity necessary for recovering the experimental observations can potentially be determined. In general, the primary goal is to reduce the time spent on computations that are unprofitable for parameter optimization. This will allow a more thorough interrogation of the parameter space, and time-intensive and high-dimensional problems, once thought intractable, might be solved.

1.2 *Parameter Optimization Framework*

A black-box parameter optimization problem can be defined in terms of an interaction between an optimization routine and an objective function [16]. This interaction is illustrated in Figure 1.1, where the data flow is shown by arrows and the essential parts are shown by the named boxes. In general, the objective function f accepts input parameters and returns results based upon its internal processes. In Figure 1.1, the internal processes of f consist of the CFD simulation and the numerical image registration. The optimization framework determines which parameter values to test based on a cached history of previous choices and their function values, initialization information, and termination criteria.

The traditional model, which is represented by the solid lines, is limited in the interaction between the objective function and the optimization framework. Without the opportunity to monitor or influence the computational process of the objective function, the cost may be too great to allow adequate exploration of the parameter space [16].

To overcome this problem, focus will be placed on the additional data flow represented by the dashed lines in Figure 1.1. The research of Magallanez [60] was based on minimizing an objective function similar to (1.1). The difference is that the objective function in [60] was optimized using only the cached history of the model parameters and the numerical image registration subprocess CPU times, labeled as “*model parameters*” and “*cputime2*”, respectively. As previously mentioned, this method implies a direct correlation in the objective function value and computational time. Model parameters may include *Reynolds* and *Prandtl* numbers and material and physical state properties in fluid flow, such as velocities, pressures, overall simulation lengths, etc.

The focus of this research extends the optimization of (1.1) to a framework that makes use of the additional data of the computational parameters λ , such as time

step Δt or grid size Δz , labeled in Figure 1.1 as “*computational parameters*”. These simulation fidelity variables control the CPU time of the CFD simulation. The use of intermediate lower fidelity simulations can possibly guide the optimization process in a way that recovers model parameters in less overall computational time while maintaining a correlation between objective function value and computational time, until higher fidelity simulations are required later in the process, which would increase the overall computation time. This allows the optimization framework to use the simulation subprocess CPU times, labeled “*cputime1*” in Figure 1.1, to choose the next set of parameter values to test that will most likely minimize the objective function f and the computational time.

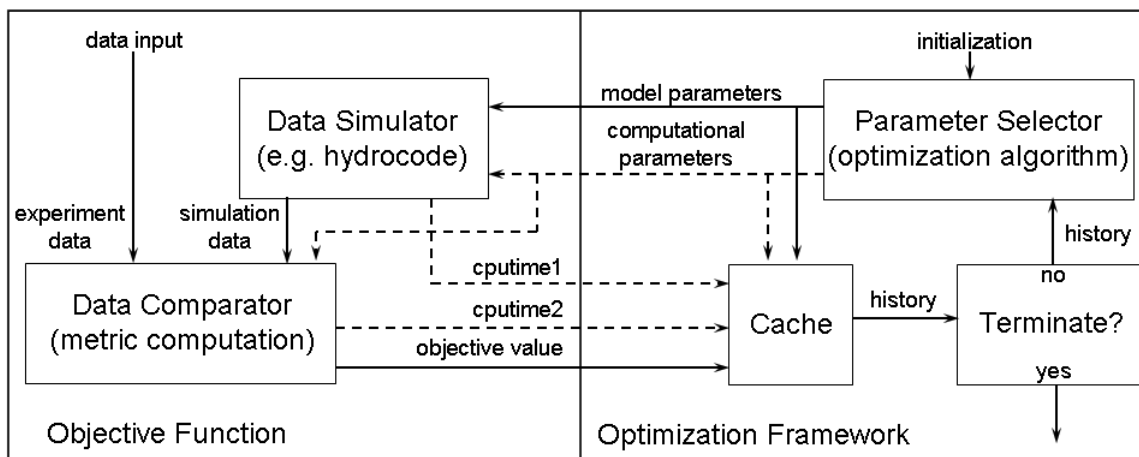


Figure 1.1 Parameter Optimization Control Diagram

These methods encourage the inexpensive parameter space search illustrated in Figure 1.2 [16]. The solid lines represent the level curves of the CPU time with $t_i < t_{i+1}$, and the dashed lines represent the level curves of the objective function with $f_i < f_{i+1}$, $i = 1, 2, \dots$, superimposed on the parameter space. The parameter space is divided into model and computational parameters, labeled p and λ , respectively, with the optimal point shown as $x^* = \{p^*, \lambda^*\}$. The simulation fidelity, controlled by the computational parameters, increases by moving to the right. A typical parameter search is illustrated by the straight dotted line from the initial point x_0 at a fixed

simulation fidelity λ to the solution \hat{x} . The approach taken here hopes to achieve the search illustrated by the curved dotted path, in which a better solution x^* is attained at lower computational cost; *i.e.*, the model parameters are recovered at the minimum simulation fidelity.

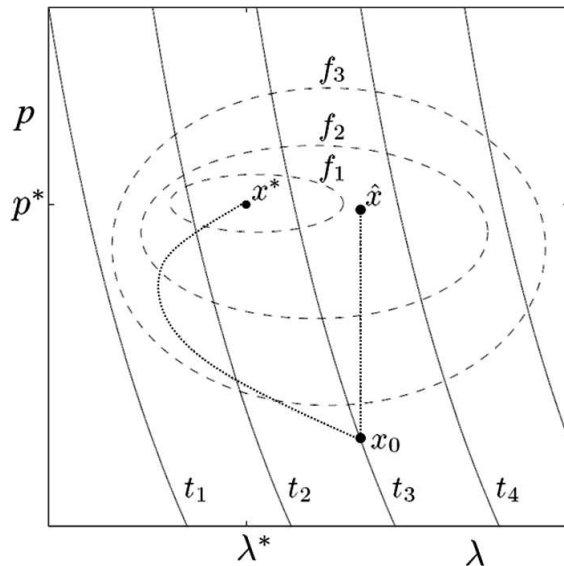


Figure 1.2 Parameter Space Search

1.2.1 Computational Fluid Dynamics Simulation

Numerical simulations are highly important when using mathematical equations to describe physical processes. These equations usually have no known analytic solution and are derived from real world observations that are valid at an infinite number of points spatially and temporally [46]. In a simulation, these equations are discretized or considered only at a finite number of selected points, so that the simulation approximates the solution to the underlying mathematical model. The accuracy of the discrete model and the computational cost of the simulation depend directly on the “fidelity” of the discretization. This leads to the need for an optimization framework for solving the discretized problem as quickly as possible while maintaining an accurate approximation of the continuous model.

Numerical simulations are important in investigating the behavior of fluid flow, where a fluid is considered to be any substance that does not have the ability to resist shear stress when at rest [46]. The appropriate mathematical model of the flow of fluid in a region $\Omega \subset \mathbb{R}^n$, $n \in \{2, 3\}$, over time $t \in [0, t_{end}]$, is characterized by a system of partial differential equations consisting of a conservation of momentum (1.2) and continuity (1.4) equation. To model temperature or heat flow, thermodynamic properties of fluids must be considered, and the conservation of energy equation (1.3) is added to the system of equations. The resulting system is known as the *Navier-Stokes equations*, defined as [46]:

$$\frac{\partial}{\partial t} \vec{u} + (\vec{u} \cdot \nabla) \vec{u} + \nabla p = \frac{1}{Re} \Delta \vec{u} + (1 - \beta T) \vec{g}, \quad (1.2)$$

$$\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T = \frac{1}{Re} \frac{1}{Pr} \Delta T + q''', \quad (1.3)$$

$$div \vec{u} = 0, \quad (1.4)$$

where $\vec{u} \in \mathbb{R}^n$ represents the velocity field, $p \in \mathbb{R}$ is the pressure in the region determined up to an additive constant, $\vec{g} \in \mathbb{R}^n$ indicates body forces such as gravity, $Re \in \mathbb{R}^+$ (*Reynolds number*) is the dimensionless ratio of inertial to viscous forces, $Pr \in \mathbb{R}^+$ (*Prandtl number*) is the relative strength of the diffusion of momentum to heat, $\beta \in \mathbb{R}$ is the coefficient of thermal expansion, and q''' is the heat source that causes changes in the temperature T , leading to variations in the fluid's density and nonlinear equations that are difficult to treat. The Laplacian (Δ) and divergence (*div*) operators used in (1.2)–(1.4) are denoted, respectively, by

$$\Delta = \sum_{i=1}^d \frac{\partial^2}{\partial x_i^2} \quad (1.5)$$

$$div = \sum_{i=1}^d \frac{\partial}{\partial x_i}. \quad (1.6)$$

There is no known analytical solution to (1.2)–(1.4). A finite difference method, using second-order central differencing for spatial discretization and first-order difference quotients for temporal discretization can be applied, resulting in a finite-dimensional problem that can be solved numerically. In the two-dimensional case, let $x = (x_{i,j}^{(k)}, y_{i,j}^{(k)})$, $u = (u_{i,j}^{(k)}, v_{i,j}^{(k)})$, and $g = (g_x, g_y)$, where $i, j = 1, \dots, m$ represent the $m \times m$ spatial points in the flow region and (k) indicates the k th iteration. Spatial variables x and y are incremented by δx and δy , respectively, and time is incremented by δt .

To fully discretize the momentum equations (1.2), central differencing is applied to the time-discretized momentum equations [46], yielding:

$$u_{i,j}^{(k+1)} = F_{i,j}^{(k)} - \frac{\delta t}{\delta x} \left(p_{(i+1),j}^{(k+1)} - p_{i,j}^{(k+1)} \right) \quad (1.7)$$

$$v_{i,j}^{(k+1)} = G_{i,j}^{(k)} - \frac{\delta t}{\delta y} \left(p_{i,(j+1)}^{(k+1)} - p_{i,j}^{(k+1)} \right), \quad (1.8)$$

where F and G are given by

$$F_{i,j}^{(k)} = u_{i,j} + \delta t \left(\frac{\Delta u_{i,j}}{Re} - \left[\frac{\partial(u^2)}{\partial x} \right]_{i,j} - \left[\frac{\partial(uv)}{\partial y} \right]_{i,j} + g_x \right) \quad (1.9)$$

$$G_{i,j}^{(k)} = v_{i,j} + \delta t \left(\frac{\Delta v_{i,j}}{Re} \left[\frac{\partial(uv)}{\partial x} \right]_{i,j} - \left[\frac{\partial(v^2)}{\partial y} \right]_{i,j} + g_y \right). \quad (1.10)$$

Equations (1.7)–(1.10) must be altered to incorporate the temperature phenomena described by (1.3), resulting in:

$$u_{i,j}^{(k+1)} = \tilde{F}_{i,j}^{(k)} - \frac{\delta t}{\delta x} \left(p_{(i+1),j}^{(k+1)} - p_{i,j}^{(k+1)} \right), \quad (1.11)$$

$$v_{i,j}^{(k+1)} = \tilde{G}_{i,j}^{(k)} - \frac{\delta t}{\delta y} \left(p_{i,(j+1)}^{(k+1)} - p_{i,j}^{(k+1)} \right), \quad (1.12)$$

$$\tilde{F}_{i,j}^{(k)} = F_{i,j}^{(k)} - \beta \frac{\delta t}{2} \left(T_{i,j}^{(n+1)} + T_{(i+1),j}^{(n+1)} \right) g_x, \quad (1.13)$$

$$\tilde{G}_{i,j}^{(k)} = G_{i,j}^{(k)} - \beta \frac{\delta t}{2} \left(T_{i,j}^{(n+1)} + T_{i,(j+1)}^{(n+1)} \right) g_y. \quad (1.14)$$

The final results are the fully discretized temperature-adapted momentum and conservation of energy equations:

$$\begin{aligned} & \frac{p_{(i+1),j}^{(k+1)} - 2p_{i,j}^{(k+1)} + p_{(i-1),j}^{(k+1)}}{(\delta x)^2} + \frac{p_{i,(j+1)}^{(k+1)} - 2p_{i,j}^{(k+1)} + p_{i,(j-1)}^{(k+1)}}{(\delta y)^2} \\ &= \frac{1}{\delta t} \left(\frac{\tilde{F}_{i,j}^{(k)} - \tilde{F}_{(i-1),j}^{(k)}}{\delta x} + \frac{\tilde{G}_{i,j}^{(k)} - \tilde{G}_{i,(j-1)}^{(k)}}{\delta y} \right) \end{aligned} \quad (1.15)$$

$$\begin{aligned} & \left[\frac{\partial T}{\partial t} \right]_{i,j}^{(n+1)} + \left[\frac{\partial(uT)}{\partial x} \right]_{i,j}^n + \left[\frac{\partial(vT)}{\partial y} \right]_{i,j}^n \\ &= \frac{1}{Re} \frac{1}{Pr} \left(\left[\frac{\partial^2 T}{\partial x^2} \right]_{i,j}^n + \left[\frac{\partial^2 T}{\partial y^2} \right]_{i,j}^n \right) + q_{i,j}''' \end{aligned} \quad (1.16)$$

Equations (1.7)–(1.16) lead to a CFD algorithm for numerically simulating fluid flow in a region. To ensure stability of the algorithm and prevent cyclic disturbances, *Courant-Friedrichs-Lewy* (CFL) stability conditions [36], must be imposed on the stepsizes to ensure that no fluid particle may travel a distance greater than the mesh spacing δx or δy in time δt [46]. The CFL conditions are implemented using the relationship,

$$\delta t = \tau \min \left(\frac{Re}{2} \left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} \right)^{-1}, \frac{Pr Re}{2} \left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} \right)^{-1}, \frac{\delta x}{|u_{max}|}, \frac{\delta y}{|v_{max}|} \right), \quad (1.17)$$

where $\tau \in [0, 1]$. The algorithm for the CFD simulation, which was also used by Magallanez [60], is summarized in Figure 1.3. Figure 1.4 depicts an image of the heat transfer flow property within the region, which is the simulation output converted to image data in both scale and property. Other flow properties of interest may include velocity, pressure, vorticity, etc. The simulation output images are used by the numerical image registration process, as will be seen in Section 1.2.2.

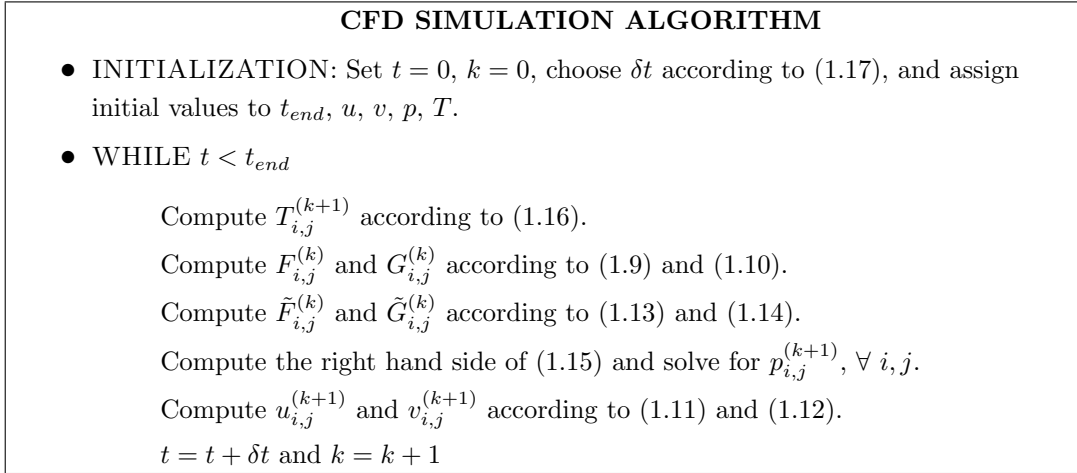


Figure 1.3 Computational Fluid Dynamics Simulation Algorithm

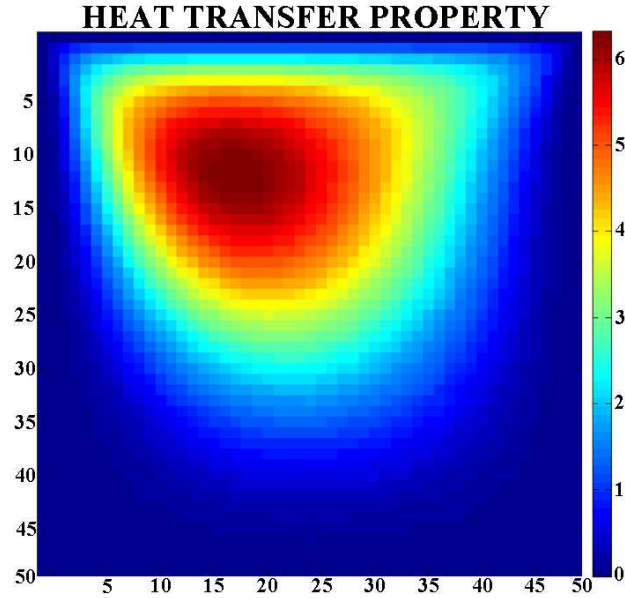


Figure 1.4 Computational Fluid Dynamics Simulation Flow Property

1.2.2 Numerical Image Registration

Numerical image registration is a fundamental problem found in many physical image processing application areas. Haber [47] and Modersitzki [62] each define the image registration problem as a means of finding a suitable spatial transformation such that the difference between a transformed template image and a reference image becomes small or the images are reasonably similar. The problem is mathematically

defined as follows: Given a *reference image* $R(x)$ created from known parameters and a *template image* $T(x)$ created from a different set of parameters, find a transformation $u : \mathbb{R}^n \rightarrow \mathbb{R}^n$ such that $T(x + u(x)) \approx R(x)$, where $x \in \Omega$ and $\Omega = [0, 1]^n$ is the region of consideration in n spatial dimensions. A visual display of this process can be seen in Figure 1.5 using the heat transfer flow property from Figure 1.4. The upper left image is created from computational and model parameters that are known. The upper right image is created using different values for the same set of parameters. The lower left image represents a transformation $u(x)$, as applied to the second image, and the final image displays the difference in the transformed and original reference image.

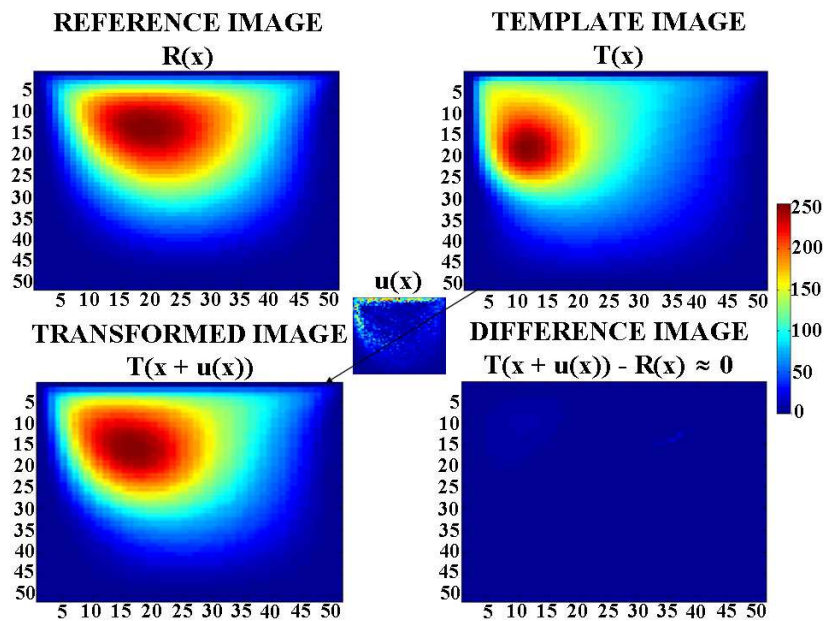


Figure 1.5 Numerical Image Registration of Simulated Flow Properties

The most intuitive way to pose the image registration problem is to choose a distance measure D and minimize the distance between R and $T(x + u(x))$ with respect to u [47, 62]. The result is the optimization problem,

$$\min_u D(T(x + u(x)) - R(x)). \quad (1.18)$$

A direct solution of (1.18) has drawbacks. The problem is difficult because small changes in input may lead to large output changes, the solution is not unique since the problem is not convex, and the transformation may not even be continuous. The remedy to these situations, suggested in [62], is to add a regularizing parameter $\alpha^{image} \in \mathbb{R}^+$ and a smoothing term S :

$$\min_u D(T(x + u(x)) - R(x)) + \alpha^{image} S(u). \quad (1.19)$$

To solve (1.19) numerically, choices must be made for the distance measure D and the smoothing term S . In defining D , consider the inner product space of squared Lebesgue-integrable functions over the Ω domain, given by

$$L_2(\Omega) = \left\{ f : \Omega \rightarrow \mathbb{R} \mid \int_{\Omega} |f(x)|^2 dx < \infty \right\}. \quad (1.20)$$

In this space, the distance measure of the difference between the transformed and reference images is defined as

$$\begin{aligned} D(u) &= \frac{1}{2} \|T(x + u(x)) - R(x)\|_{L_2(\Omega)} \\ &= \frac{1}{2} \int_{\Omega} (T(x + u(x)) - R(x))^2 dx. \end{aligned} \quad (1.21)$$

where $\|\cdot\|_{L_2(\Omega)}$ is the norm induced by (1.20). The smoothing term and regularizer are defined by the curvature regularizer

$$S[u] = \frac{1}{2} \sum_{l=1}^n \int_{\Omega} (\Delta u_l)^2 dx. \quad (1.22)$$

To use these building blocks, the Gâteaux derivative (a generalization of the directional derivative in differential calculus) of the distance measure D and the

smoothing term S must exist. This derivative applied to (1.21–1.22) yields

$$\begin{aligned} d[D(u)] &= \int_{\Omega} (T(x + u(x)) - R(x)) \cdot \nabla T(x + u(x)) dx \\ &= \int_{\Omega} f(x, u(x)) dx \end{aligned} \tag{1.23}$$

$$d[S(u)] = \int_{\Omega} A[u](x) dx, \tag{1.24}$$

where $f(x, u(x))$ is the force measurement associated with transforming image pixels over a distance D , and $A[u] = \Delta^2 u$ is a partial differential operator associated with the smoothing term S . The computed force f is used to transform the template image $T(x)$ to the reference image $R(x)$, thus defining the transformation $u(x)$. The approach used by Haber [47] and Modersitzki [62] applies the Euler-Lagrange equations to (1.23)–(1.24). This yields the system of nonlinear partial differential equations,

$$A[u](x) - f(x, u(x)) = 0, \quad \forall x \in \Omega, \tag{1.25}$$

from which a fixed-point iteration scheme is developed to by-pass the nonlinearity of (1.25):

$$A[u^{k+1}](x) - f(x, u^k(x)) = 0. \tag{1.26}$$

The iteration scheme (1.26) leads to a numerical algorithm for image registration, which has two problems that must be solved: the computation of the force used for the transformation and the numerical solution of the partial differential equations (1.25). The numerical image registration algorithm, which was also used by Magallanez [60], is summarized in Figure 1.6.

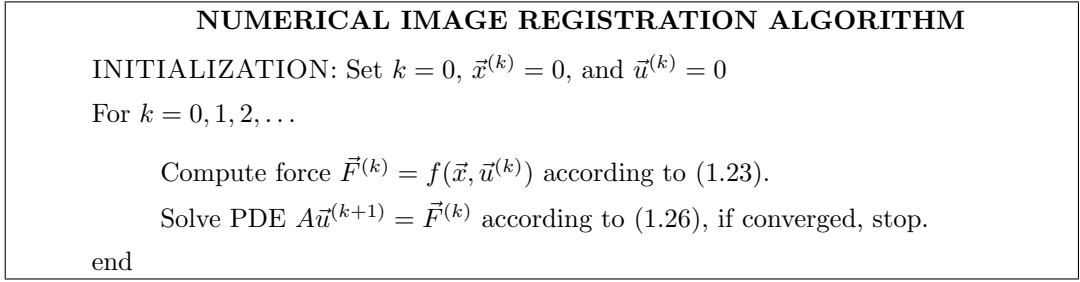


Figure 1.6 Numerical Image Registration Algorithm

1.2.3 Dual Process Objective Function

As previously stated, the objective function in (1.1) consists of the fluid flow simulation and the image registration processes given in Figures 1.3 and 1.6, respectively. The objective function takes model parameters, such as *Reynolds* number, *Prandtl* number, velocities, etc., and computational parameters, such as grid size, step size, etc., associated with fluid flow, and uses them as inputs within the CFD simulation to create images of certain flow properties, such as the heat transfer flow property illustrated in Figure 1.4. This template image is then compared to a reference image for which the input parameters are known. The goal is to recover the model parameters of the reference image and possibly the minimal set of values for the computational parameters. As discussed in Section 1.2.2, the image registration optimization problem (1.19) requires a measure of distance to represent the differences in the two images, as seen in the process of Figure 1.5. This implies the optimal transformation $u^*(x)$, which is the solution to (1.19), can be used to define the objective function in the form of (1.1). This research (and that of Magallanez [60]) uses the idea of displacement fields as applied to the visual pixels of the image. Given that $u^*(x)$ represents the optimal displacement or transformation fields, and \bar{u}^* represents the average displacement field of the image pixels, the objective function can be defined with respect to the induced L_2 -norm of the difference of these two fields:

$$f(x) = \|u^*(x) - \bar{u}^*\|_{L_2(\Omega)}. \tag{1.27}$$

If the images are approximately the same, the value for (1.27) will be close to zero. This implies that the model parameters used to create the template image are similar to the actual model parameters of the reference image. This does not necessarily imply that the computational parameters have to be the same between the two images. For example, allowing a lower fidelity model parameter, such as grid size Δz , decreases both the fidelity and computational time of the fluid simulation. However, it is possible that the images are approximately the same at this lower fidelity. Therefore, it takes less computational time for the fluid simulation, and fewer iterations, or less time for the numerical image registration algorithm to transform the template image into the reference image. If the images differ greatly, which implies that either the fidelity of the computational parameters is too low, or the difference in model parameters between the template and reference images is large, then it takes more time to transform, and the transformation $u^*(x)$ computed by the image registration may not yield a zero objective function value. This suggests that a correlation may exist between objective function values and their associated computational times, in that computational time decreases as objective function values decrease.

As with [60], this correlation does exist to a certain extent. However, it is possible to have a template image that differs significantly from the reference image, with two other outcomes: an increased objective function value with a decreased computational time, or a decreased objective function value with an increased computational time. This is due to the forces computed in (1.23) that are used to displace the pixels of the flow property image. For the first case, the forces can be expended quickly creating a divergent template image (that differs significantly from the reference image), which causes the image registration algorithm to prematurely exit, resulting in an increased objective function value with artificially low computational time. For the second case, the forces are slowly dissipated, resulting in a very high computational time and a template image still far away from the reference image. Remedies for these cases are discussed in Chapters 2 and 3.

1.3 Purpose

The goal of this research is to develop and implement an optimization framework for solving a class of optimization problems that require expensive function evaluations while reducing the time spent on unprofitable computations. The framework makes careful use of computing resources through the careful employment of multiple surrogate functions and the utilization of direct search methods to determine the optimal parameters. Surrogate functions, which require much less computation time, guide the direct search methods in the optimization framework, such that the optimal point of the surrogate is easily found and is a potential incumbent of the objective function. Direct search methods are used to solve the surrogate problem as well as (1.1), due to the robustness of the method and the lack of derivative information for the functions.

1.4 Overview

The remainder of this thesis is laid out as follows. Chapter 2 discusses the results of the previous work of Magallanez [60] and outlines the relevant literature on surrogate functions and direct search methods. Chapter 3 describes the methodology that was developed for using both methods and selectively employing multiple surrogates. Chapter 4 describes the implementation of the method on several test problems and presents the results. Chapter 5 finishes with final conclusions and possible avenues for future research.

2. Relevant Literature

This chapter reviews the relevant literature related to the development of the methods to solve the optimization problem (1.1). The first section defines surrogate functions, including a development of their structure and applicability to computationally expensive problems through the use of a surrogate-based optimization framework. The subsequent sections describes certain “derivative-free” direct search methods and their use in solving nonlinear black-box optimization problems. The final section discusses the research results of Magallanez [60].

2.1 Optimization Using Surrogate Functions

One of the most challenging issues in using standard optimization techniques in real world applications is that the objective function can require significant amounts of computational time [78]. Expensive functions are typically handled by optimizing some type of less expensive approximation model or surrogate function. In fact, this idea was the impetus behind response surface methodologies (RSM) introduced by Box and Wilson [26]. However, in RSM the goal is to optimize the approximation model for which the coefficients of the approximating polynomial are estimated but not known. This can lead to a value that differs significantly from the true optimal point. Direct optimization that employs approximation models, or surrogates, originated in the 1970s with the work of Smith and Miura [77], and has proven popular for real world applications [41]. However, one problem with applying surrogates without evaluating the objective function at a sufficient number of points is the possibility of failure because there is not enough information to build an adequate model. A method of mitigating this problem is to build the surrogate with previously evaluated points, and during the optimization process, adapt the surrogate by sampling the true function at selected points. Supporting mathematical theory has been developed for this type of approximation-based optimization method. Alexandrov *et*

al. [13] and Booker *et al.* [23], inspired by the ideas of Dennis and Torczon [39], use the terms *surrogate-based optimization* and *surrogate management framework* to describe these methods for solving optimization problems using a surrogates.

Surrogates used within this type of optimization framework are of two types: simplified physics models that describe the behavior of the physical system at all points or functional data fit models that are purely mathematical constructs for modeling the behavior of the true function through an approximating function.

2.1.1 *Simplified Physics Models*

A simplified physics surrogate is a lower fidelity physics-based model (with reduced computational cost) that is used in conjunction with, or in place of, the costly higher fidelity model. As discussed in [41], simplified physics surrogates can be categorized by the methods used to reduce the fidelity and may include any of the following: coarser discretizations, relaxed residual tolerances, omission of certain modeled physical properties, or reduction in dimensionality.

Examples of one-dimensional simplified physics surrogates are given by the solid lines in Figure 2.1, where the open circles and dashed line represent the sampled values and an unknown objective function respectively. As noted in [78], a disadvantage of using this type of surrogate is that it requires knowledge of the behavior and complexity of the physical system, thus implying that the use of simplified physics surrogates is problem-specific.

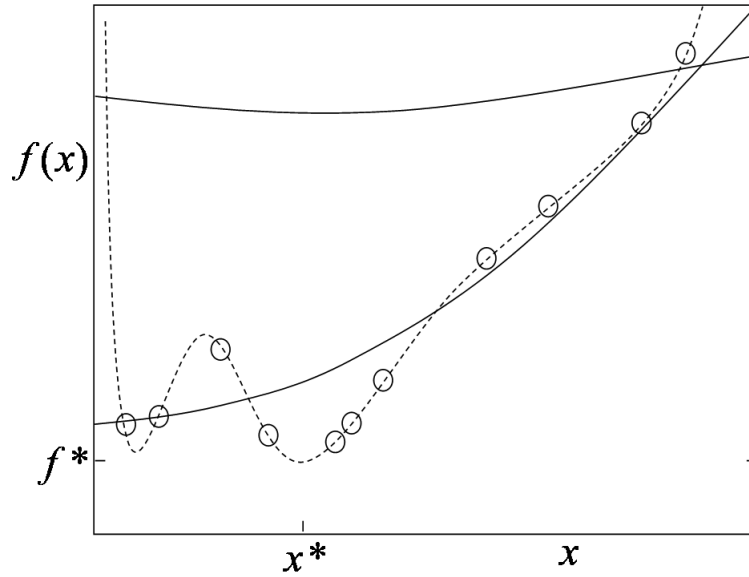


Figure 2.1 Simplified Physics Surrogate

Following the methods of Alexandrov *et al.* [13] and Robinson *et al.* [72], a simplified physics model can be used to derive the surrogate under conditions where the model is defined over the same design space or a lower dimension design space. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ represent the objective function to be minimized and $x \in \mathbb{R}^n$ is the vector of n design variables that describe the design. A low fidelity model for $f(x)$ is denoted by $g(\tilde{x})$ where $\tilde{x} \in \mathbb{R}^{\tilde{n}}$ is the low fidelity design vector of dimension $\tilde{n} \leq n$ (*i.e.*, the model is defined over the same or a lower dimensional space than the objective function). The low fidelity model is used to derive the surrogate model $\tilde{f}(x)$, which is defined over the same design space as the objective function. An appropriate simplified physics surrogate $\tilde{f}(x)$ that approximates $f(x)$ is at least first-order accurate at the center of the design region [13]. A consistent surrogate model can be constructed from the low fidelity model $g(\tilde{x})$ through the use of additive and multiplicative corrections as well as space mapping transformations for models defined over a lower dimensional space. A corrected surrogate model for the additive

and multiplicative cases is given by

$$\tilde{f}(x) = g(\tilde{x}) + \alpha(x), \quad (2.1)$$

$$\tilde{f}(x) = g(\tilde{x})\beta(x), \quad (2.2)$$

where α and β are defined as approximations to the correction functions used in [72].

If the low fidelity model $g(\tilde{x})$ is defined over a lower dimensional space then a space mapping transformation is required. As defined in [21, 72], space mapping is a method of mapping between models of different dimensionality or fidelity. Let P denote the space mapping function between the high and low fidelity models. Bandler *et al.* [21] provide a detailed method and description for finding optimal space mapping functions. For the types of problems considered here, initial points used by $g(\tilde{x})$ are those at which the objective function values have already been computed. The transformation P is a mapping such that

$$\tilde{x} = P(x), \quad (2.3)$$

and the combination of the additive (2.1) and multiplicative corrected surrogate (2.2) with the space mapping transformation (2.3), respectively, results in the corrected surrogate models for the two cases:

$$\tilde{f}(x) = g(P(x)) + \alpha(x) \quad (2.4)$$

$$\tilde{f}(x) = g(P(x))\beta(x). \quad (2.5)$$

2.1.2 *Functional Data Fit Models: Design and Analysis of Computer Experiments (DACE)*

Complex computer models have become an increasingly popular method of investigating many scientific phenomena [74, 75]. A feature of using computers to perform experiments is that the output is deterministic, given that the experiment

is conducted using the same input parameters. However, these computer models are usually computationally expensive; therefore, a less expensive method of predicting their output is desirable. One method of prediction is a functional data fit model, which is a purely mathematical construct that models the behavior of the true function through an approximating function and makes no assumptions based upon physics of the objective function. An example of a functional data fit model is a “Design and Analysis of Computer Experiments” (DACE) surrogate that represents the trends of a response over the range of the design variables [74, 75]. DACE surrogates are defined overall by an initial design of experiments and a kriging surrogate.

Design of experiments can be divided into two types: classical and modern. Both techniques share the common goal of extracting as much information as possible from a limited set of laboratory or computer experiments [45]. Classical design of experiments, such as central composite design and Box-Behnken design, have a rich history of statistical and mathematical development along with practical application in scientific and engineering studies [63]. Modern design of experiments methods aim to generate in the design space a “space-filling” set of points from which to sample the objective function. Examples include Latin Hypercube sampling [80, 81], Orthogonal Arrays [69], and nearly uniform designs [59].

Kriging surrogates are motivated by the requirement to sample fewer points and have been extensively used for approximating deterministic computer models used in a variety of applications, such as structural optimization, multidisciplinary design, aerospace engineering, and mechanical engineering [23, 24, 61, 79]. In kriging, the deterministic response $y(x)$ of a computer model is treated as a realization of a random function $\hat{y}(x)$ that consists of a generalized linear regression model and a random process accounting for the correlation in the residuals between the regression model and actual objective function values. This allows for the construction of an

approximating model that can interpolate the observed or known data points of the actual objective function [61].

The structure and development of the kriging surrogate is given following the derivations of [56, 61, 64, 74]. The mathematical form of a kriging surrogate has two components:

$$\hat{y}(x) = \sum_{j=1}^n \beta_j f_j(x) + Z(x) = \beta^T f(x) + Z(x), \quad (2.6)$$

where $\beta = [\beta_1, \beta_2, \dots, \beta_n] \in \mathbb{R}^n$ and $f = [f_1, f_2, \dots, f_n]$. For clarification purposes, unless specifically referenced, f in this notation is not the same $f(x)$ being optimized in (1.1). The $\beta^T f(x)$ term is a linear regression function modeling any trends over the domain, and $Z(x)$ is the realization of a stationary Gaussian random function with zero mean and a covariance between $Z(w)$ and $Z(x)$ of

$$V(w, x) = E[Z(w)Z(x)] = \sigma^2 R(\theta, w, x) \quad (2.7)$$

that defines the differences between the actual objective function values of $f(x)$ and the interpolated data. The value of σ^2 is the process variance of the response, and $R(\theta, w, x)$ is the spatial correlation function of the kriging model. The correlation function R with parameter θ is used to control the smoothness of the surrogate, influence of nearby points, and differentiability of the response surface. For the purpose of the derivations, the deterministic response at an untried point \tilde{x} can be expressed as:

$$y(\tilde{x}) = \sum_{j=1}^n \beta_j f_j(\tilde{x}) + Z(\tilde{x}), \quad (2.8)$$

where $Z(\tilde{x})$ has a mean of zero and a covariance of

$$V(\tilde{x}, \tilde{x}) = E[(Z(\tilde{x}))^2] = \sigma^2 \quad (2.9)$$

that behaves as an approximation error. Under the assumption that β is chosen correctly, the error will behave as noise [56].

The kriging predictor is constructed using a regression model that is a linear combination of p functions $f_j : \mathbb{R}^n \rightarrow \mathbb{R}$ (see by Lophaven *et al.* [56]) and denoted by:

$$\begin{aligned}\hat{F}(\beta, x) &= \beta_1 f_1(x) + \cdots + \beta_p f_p(x) \\ &= [f_1(x) \cdots f_p(x)]^T \beta \\ &\equiv f(x)^T \beta,\end{aligned}\tag{2.10}$$

which allows the kriging surrogate (2.6) and the deterministic response (2.8) to be rewritten, respectively, as

$$\hat{y}(x) = \hat{F}(\beta, x) + Z(x) \equiv f(x)^T \beta + Z(x)\tag{2.11}$$

$$y(\tilde{x}) = \hat{F}(\beta, \tilde{x}) + Z(\tilde{x}) \equiv f(\tilde{x})^T \beta + Z(\tilde{x}).\tag{2.12}$$

Given a set of known design points $\{x_i\}_{i=1}^k \subset \mathbb{R}^n$ and their responses $Y = \{y_i\}_{i=1}^k \subset \mathbb{R}^m$, denote the design matrix F with $[F]_{ij} = f_j(x_i)$ by,

$$F = [f(x_1) \cdots f(x_k)]^T\tag{2.13}$$

where $f(x)$ is defined by (2.10). The spatial correlation function R in (2.7) is the matrix of random process correlations between $Z(x_i)$ and $Z(x_j)$ at the design points denoted by,

$$R_{ij} = R(\theta, x_i, x_j), \quad i, j = 1, \dots, k.\tag{2.14}$$

Denote the vector of correlations between $Z(x_i)$ at the design sites and $Z(\tilde{x})$ at an untried point \tilde{x} by

$$r(\tilde{x}) = (R(\theta, x_1, \tilde{x}) \cdots R(\theta, x_m, \tilde{x}))^T. \quad (2.15)$$

The linear predictor of the true value $Y = y(x)$ at an untried point \tilde{x} is given by

$$\hat{y}(\tilde{x}) = c(\tilde{x})^T Y, \quad (2.16)$$

and the mean square error (MSE) of this predictor is computed over the random process. The best linear unbiased predictor is found by choosing the coefficients $c(\tilde{x}) \in \mathbb{R}^m$ that minimize the MSE. The error between (2.16) and (2.8) is

$$\hat{y}(\tilde{x}) - y(\tilde{x}) = c(\tilde{x})^T Y - y(\tilde{x}). \quad (2.17)$$

Substituting (2.11)–(2.13) into the right hand side of (2.17) yields:

$$\begin{aligned} \hat{y}(\tilde{x}) - y(\tilde{x}) &= c(\tilde{x})^T (F\beta + Z(x)) - (f(\tilde{x})^T \beta + Z(\tilde{x})) \\ &= c(\tilde{x})^T Z(x) - Z(\tilde{x}) + (c(\tilde{x})^T F - f(\tilde{x}))\beta. \end{aligned} \quad (2.18)$$

In order to ensure the kriging predictor is unbiased the unbiasedness constraint

$$c(\tilde{x})^T F - f(\tilde{x}) = 0 \quad (2.19)$$

is added. This constraint and (2.18) imply that the MSE of (2.16) is derived as

$$\begin{aligned} MSE[\hat{y}(\tilde{x})] &= E[(\hat{y}(\tilde{x}) - y(\tilde{x}))^2] \\ &= E[(c(\tilde{x})^T Z(x) - Z(\tilde{x}))^2] \\ &= E[Z(\tilde{x})^2 + c(\tilde{x})^T Z(x) Z(x)^T c(\tilde{x}) - 2c(\tilde{x})^T Z(\tilde{x}) Z(x)]. \end{aligned} \quad (2.20)$$

Using the covariance definitions (2.7) and (2.9), as well as the spatial correlation function definitions (2.14) and (2.15) to simplify (2.20), results in:

$$\begin{aligned} E[Z(\tilde{x})^2] &= \sigma^2, \\ E[Z(x)Z(x)^T] &= \sigma^2 R(\theta, x, x) = \sigma^2 R, \\ E[Z(\tilde{x})Z(x)] &= \sigma^2 R(\theta, \tilde{x}, x) = \sigma^2 r, \end{aligned}$$

which simplifies (2.20) further, yielding

$$MSE[\hat{y}(\tilde{x})] = \sigma^2(1 + c(\tilde{x})^T R c(\tilde{x}) - 2c(\tilde{x})^T r). \quad (2.21)$$

To find the coefficients $c(\tilde{x})$ that minimize (2.21), subject to the unbiasedness constraint (2.19), the Lagrangian function is formulated with Lagrange multipliers $\lambda(\tilde{x}) \in \mathbb{R}^m$ as

$$L(c, \lambda) = \sigma^2(1 + c(\tilde{x})^T R c(\tilde{x}) - 2c(\tilde{x})^T r) - \lambda(\tilde{x})^T (c(\tilde{x})^T F - f(\tilde{x})), \quad (2.22)$$

and taking the gradient of L with respect to $c(\tilde{x})$ and $\lambda(\tilde{x})$ yields the first order necessary conditions,

$$\begin{aligned} \nabla_c L(c, \lambda) &= 2\sigma^2(Rc(\tilde{x}) - r) - F\lambda(\tilde{x}) = 0 \\ \nabla_\lambda L(c, \lambda) &= Fc(\tilde{x}) - f(\tilde{x}) = 0 \end{aligned}$$

which yields

$$\Rightarrow Rc(\tilde{x}) + F\tilde{\lambda}(\tilde{x}) = r \quad (2.23)$$

$$\Rightarrow Fc(\tilde{x}) = f(\tilde{x}), \quad (2.24)$$

where $\tilde{\lambda}(\tilde{x})$ of (2.23) is defined by

$$\tilde{\lambda}(\tilde{x}) = -\frac{\lambda(\tilde{x})}{2\sigma^2}.$$

Equations (2.23) and (2.24) yield the system of equations,

$$\begin{bmatrix} R & F \\ F^T & 0 \end{bmatrix} \begin{bmatrix} c(\tilde{x}) \\ \tilde{\lambda}(\tilde{x}) \end{bmatrix} = \begin{bmatrix} r \\ f(\tilde{x}) \end{bmatrix}, \quad (2.25)$$

whose solution is

$$\tilde{\lambda}(\tilde{x}) = (F^T R^{-1} F)^{-1} (F^T R^{-1} r - f(\tilde{x})) \quad (2.26)$$

$$c(\tilde{x}) = R^{-1} (r - F \tilde{\lambda}(\tilde{x})). \quad (2.27)$$

Substituting the solutions (2.26)–(2.27) into (2.16) yields the predictor

$$\begin{aligned} \hat{y}(\tilde{x}) &= c(\tilde{x})^T Y, \\ &= (r - F \tilde{\lambda}(\tilde{x}))^T R^{-1} Y \\ &= r^T R^{-1} Y - (F^T R^{-1} r - f(\tilde{x}))^T (F^T R^{-1} F)^{-1} F^T R^{-1} Y \\ &= r^T R^{-1} Y - (F^T R^{-1} r - f(\tilde{x}))^T \beta^* \\ &= f(\tilde{x})^T \beta^* + r^T R^{-1} (Y - F \beta^*) \\ &= f(\tilde{x})^T \beta^* + r(\tilde{x})^T \gamma^*, \end{aligned} \quad (2.28)$$

where β^* and γ^* are the generalized least squares solutions, given by

$$\beta^* = (F^T R^{-1} F)^{-1} F^T R^{-1} Y \quad (2.29)$$

$$\gamma^* = R^{-1} (Y - F \beta^*) \equiv \sigma^2 (1 + c(\tilde{x})^T R c(\tilde{x}) - 2c(\tilde{x})^T r). \quad (2.30)$$

Regression models used in (2.6) are polynomials of order 0, 1, and 2 [56]. More specifically:

$$\begin{aligned}
\text{Constant (order 0) :} & \quad f_1(x) = 1 \\
\text{Linear (order 1) :} & \quad f_1(x) = 1, f_2(x) = x_1, \dots, f_{n+1}(x) = x_n \\
\text{Quadratic (order 2) :} & \quad f_1(x) = 1, f_2(x) = x_1, \dots, f_{n+1}(x) = x_n, \\
& \quad f_{n+2}(x) = x_1^2, \dots, f_{2n+1} = x_1 x_n, f_{2n+2}(x) = x_2^2, \\
& \quad \dots, f_{3n} = x_2 x_n, \dots, f_p(x) = x_n^2.
\end{aligned} \tag{2.31}$$

The correlation models for (2.7) are typically restricted [56] to the form:

$$R(\theta, w, x) = \prod_{j=1}^n R_j(\theta, w_j - x_j) = \prod_{j=1}^n R_j(\theta_j, d_j), \tag{2.32}$$

where $d_j = w_j - x_j$, $j = 1, 2, \dots, n$, and $R_j(\theta_j, d_j)$ in (2.32) is usually defined by one of the following functions:

$$\begin{aligned}
\text{Exponential} & : \exp(-\theta_j |d_j|) \\
\text{General Exponential} & : \exp(-\theta_j |d_j|^{\theta_{n+1}}), \quad 0 < \theta_{n+1} \leq 2 \\
\text{Gaussian} & : \exp(-\theta_j d_j^2) \\
\text{Linear} & : \max\{0, 1 - \theta_j |d_j|\} \\
\text{Spherical} & : 1 - 1.5\xi_j + 0.5\xi_j^3, \quad \xi_j = \min\{1, \theta_j |d_j|\} \\
\text{Cubic} & : 1 - 3\xi_j^2 + 2\xi_j^3, \quad \xi_j = \min\{1, \theta_j |d_j|\} \\
\text{Spline} & : \varsigma(\xi_j) = \left\{ \begin{array}{ll} 1 - 15\xi_j^2 + 30\xi_j^3, & 0 \leq \xi_j \leq 0.2 \\ 1.25(1 - \xi_j)^3, & 0.2 < \xi_j < 1 \\ 0, & 1 \leq \xi_j. \end{array} \right\}, \quad \xi_j = \theta_j |d_j|.
\end{aligned} \tag{2.33}$$

The choice of the regression polynomial order, correlation model, and θ can significantly affect the quality of the surrogate. The correlation functions given in

(2.33) can be separated into two groups, one containing functions with parabolic behavior near the origin (*Gaussian*, *Cubic*, and *Spline*), and the other containing functions with linear behavior near the origin (*Exponential*, *Linear*, and *Spherical*). The choice of correlation function should be motivated by the underlying phenomena [56]. If the function representing the underlying physical phenomena is continuously differentiable, it will likely show parabolic behavior at the origin, implying a *Gaussian*, *Cubic*, or *Spline* correlation model should be used. Functions showing linear behavior at the origin, imply an *Exponential*, *General Exponential*, *Linear*, or *Spherical* correlation may perform better. As noted by Lophaven *et al.* [56], phenomena are often anisotropic, meaning that different correlations are identified in different directions; *i.e.*, the behavior of the response surface of the kriging functions may differ between directions. Allowing for different parameters θ_j , $j = 1, 2, \dots, n$, is accounted for in (2.33).

The values of $\theta_j \in [0, +\infty)$, $j = 1, 2, \dots, n$, are typically optimized, based on data sites, responses, and user-specified or user-computed lower and upper bounds for each θ_j . The optimal values of θ are found by solving

$$\min_{\theta} |R|^{1/m} \hat{\sigma}^2, \quad (2.34)$$

for the maximum likelihood estimator θ^* , where the corresponding maximum likelihood estimate of the variance is

$$\hat{\sigma}^2 = \frac{1}{m} (Y - F\beta^*)^T R^{-1} (Y - F\beta^*). \quad (2.35)$$

In practice, the unconstrained optimization problem in (2.34) needs lower and upper bounds on θ to avoid excessive computation and to prevent undesirable choices of θ . These bounds can be estimated by an iterative process using the correlation model (2.32) [10]. Lophaven *et al.* [57, 58] use a modified version of the pattern search method of Hooke and Jeeves [49] to compute optimal values of θ . The optimization

only requires an initial value of θ ; however, providing upper and lower bounds can prevent θ from converging to a poor local minimum [57].

Figure 2.2 shows DACE response surfaces (using the DACE MATLAB[®] software [56]) for an example problem, using an initial guess of [10; 10] with lower bounds $\theta_l = [0.1; 0.1]$ and upper bounds of $\theta_u = [20; 20]$. The kriging model used a 2nd order regression polynomial with *Exponential* and *Gaussian* correlation models.

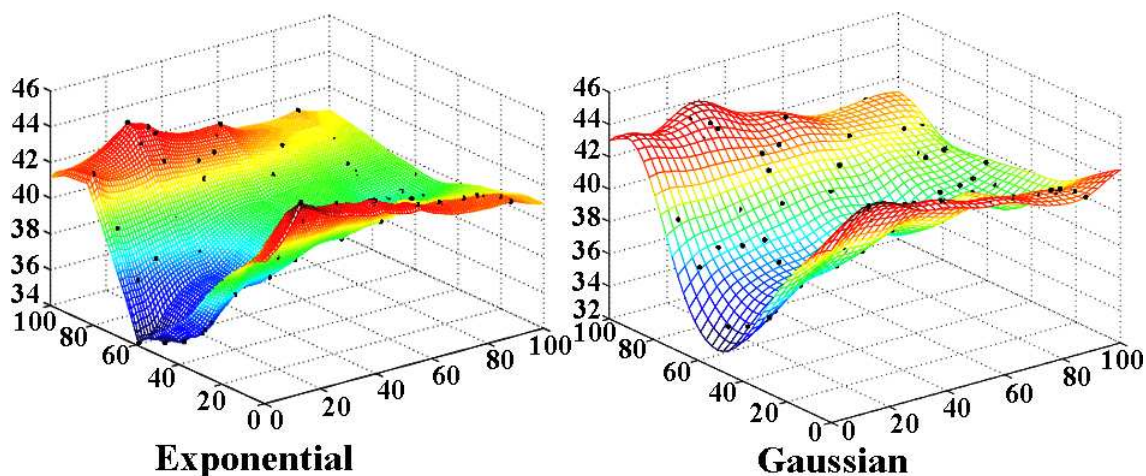


Figure 2.2 Response Surfaces of DACE Surrogates

2.1.3 Functional Data Fit Models: Radial Basis Function (RBF) Surrogates

When the functions that are approximated depend on many variables/parameters and are defined by many data points scattered throughout the domain, *Radial Basis Functions* (RBFs) are a well suited approximation approach [31]. RBFs are real-valued functions that can be used to interpolate data and approximate nonlinear functions. Following [32, 31, 71, 83], to approximate a real-valued function $f(x)$ by some interpolating model $s(x)$, given n distinct points $x_1, \dots, x_n \in \mathbb{R}^n$ with known

function values $f(x_1), \dots, f(x_n)$, the RBF interpolant is of the form:

$$s(x) = p(x) + \sum_{i=1}^n \lambda_i \phi(\|x - x_i\|), \quad x \in \mathbb{R}^n, \quad (2.36)$$

where $p(x) \in \Pi_m^n$ is the space of polynomials in n variables of degree at most m , $\lambda_i \in \mathbb{R}$ for $i = 1, \dots, n$ are real-valued weights, $\|\cdot\|$ is the Euclidean norm, and $\phi: \mathbb{R}^+ \rightarrow \mathbb{R}$ is the *basis function* [32]. In (2.36), $\phi(x) = \phi(\|x - x_i\|)$ is a measure of the distance from x to x_i . Therefore, an RBF can be thought of more technically as a weighted sum of translations of a radially symmetric basis function, augmented by a polynomial term [32].

As suggested in [66] and [71], there are various forms of the basis function, including the following:

$$\textit{Surface Splines} : \phi(r) = r^k, \quad k \in \mathbb{N}, \quad k \text{ is odd} \quad (2.37)$$

$$\phi(r) = r^k \log r, \quad k \in \mathbb{N}, \quad k \text{ is even} \quad (2.38)$$

$$\textit{Multiquadric} : \phi(r) = (r^2 + \gamma^2)^k, \quad k > 0, \quad k \notin \mathbb{N} \quad (2.39)$$

$$\textit{Inverse Multiquadric} : \phi(r) = (r^2 + \gamma^2)^k, \quad k < 0, \quad k \notin \mathbb{N} \quad (2.40)$$

$$\textit{Gaussian} : \phi(r) = e^{-\gamma r^2}, \quad (2.41)$$

where $r \geq 0$ is the radius (or distance) from the origin and γ is a positive scalar. Special cases of (2.37) include the *Bi-harmonic* and the *Tri-harmonic (Cubic Spline)* functions, where $k = 1$ and $k = 3$, respectively. A special case of (2.38) is the *Thin-plate Spline* with $k = 2$. Different advantages exist for each of these forms. The *Bi-harmonic* and the *Tri-harmonic* are good choices for fitting functions of three variables, the *Thin-plate Spline* is a good choice for fitting smooth functions of two variables, and the *Multiquadric* is used for various applications, such as fitting topographical data [32].

Regis and Shoemaker [71] and Carr *et al.* [32] describe how the RBF interpolant $s(x)$ is found by solving for the coefficients $c = (c_1, \dots, c_m)$ of the polynomial $p(x)$ and the weights λ_i in the following derivations. A basis function is chosen from one of the forms (2.37)–(2.41), and matrices $\hat{\phi} \in \mathbb{R}^{n \times n}$ and $\Gamma \in \mathbb{R}^{n \times m}$ are defined by

$$\hat{\phi}_{ij} = \phi(\|x_i - x_j\|), \quad i, j = 1, \dots, n \quad (2.42)$$

$$\Gamma_{ij} = p_j(x_i), \quad i = 1, \dots, n, j = 1, \dots, m. \quad (2.43)$$

For $f(x_1), \dots, f(x_n)$, the weights $\lambda_i, i = 1, 2, \dots, n$, are sought such that the RBF satisfies

$$s(x_i) = f(x_i), \quad i = 1, \dots, n. \quad (2.44)$$

This yields a linear system with more parameters than data; namely,

$$\sum_{j=1}^n \lambda_j p(x_j) = 0, \quad (2.45)$$

where the weights $\lambda_j, j = 1, \dots, n$, are imposed to maintain orthogonality and $p(x_j)$ is a polynomial of n variables of degree at most m . In [71], (2.42)–(2.45) are used to find the RBF that interpolates $(x_1, f(x_1)), \dots, (x_n, f(x_n))$ by solving the system,

$$\begin{bmatrix} \hat{\phi} & \Gamma \\ \Gamma^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ c \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}, \quad (2.46)$$

where $F = (f(x_1), \dots, f(x_n))^T$, $\lambda = (\lambda_1, \dots, \lambda_n)^T \in \mathbb{R}^n$, and $c = (c_1, \dots, c_m)^T \in \mathbb{R}^m$.

The value left undetermined is the scalar γ of the various basis functions. Franke [43] found in his work with the *Multiquadric* (2.39) basis function, that when γ was chosen to be close to the average distances between design sites, the interpolation in two dimensions was more accurate. Nielsen *et al.* [66] suggests using a value of $\gamma = 1$ in all basis functions; however, as the distances between design sites become

large, the value of γ affects accuracy minimally. Buhman *et al.* [30] give error estimates showing that accuracy increases as γ gets larger. Powell [70] shows that the interpolants converge uniformly as $\gamma \rightarrow \infty$ when the distances between design sites form an evenly spaced grid. For many basis functions, it would seem advantageous to choose large values of γ . However, Baxter [22] shows that if the distances form a finite evenly spaced grid, then the smallest eigenvalues of (2.42) decrease exponentially as $\gamma \rightarrow \infty$. This implies that the matrix $\hat{\phi}$ in (2.46) can become ill-conditioned, and “in general, there is no best way to choose a value of γ for the basis functions” [22].

An example of the response surface of an RBF surrogate with $n = 20$, using a *Multiquadric* basis function and 2nd order quadratic polynomial terms can be seen in Figure 2.3.

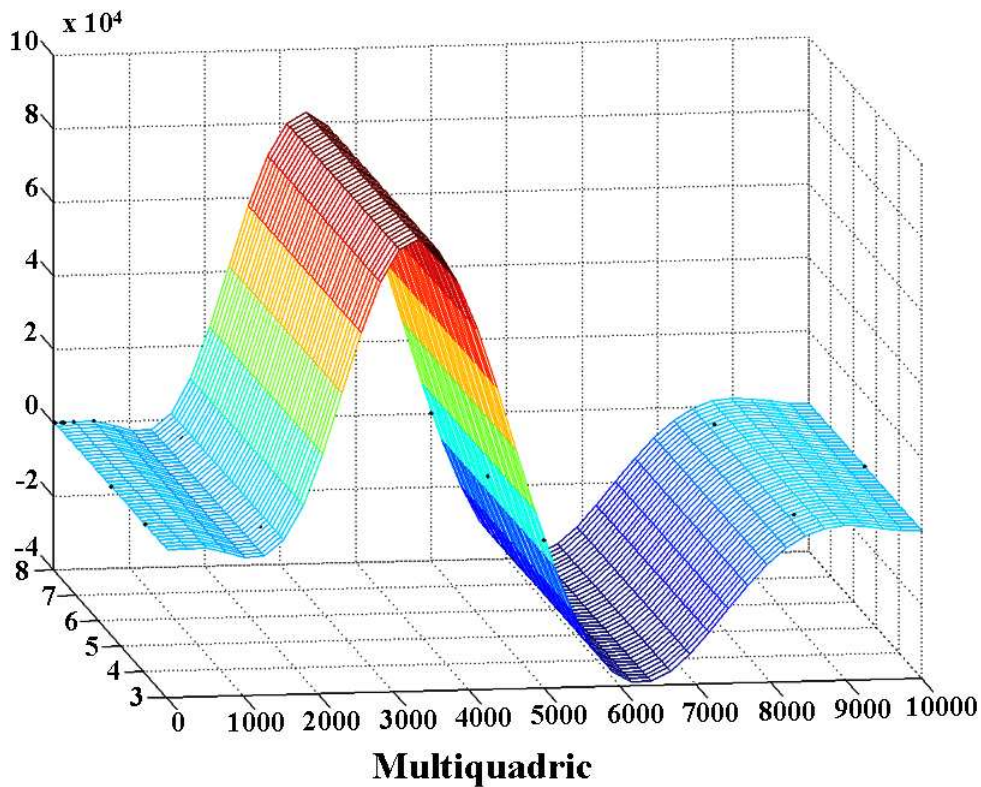


Figure 2.3 Response Surface of an RBF Surrogate

2.1.4 Surrogate Management Framework

The use of approximation models (*surrogates*) for optimizing expensive computer simulations has become a standard in engineering practice to aid in the guidance of design processes [23, 39]. These models help reduce the number of computationally expensive function evaluations. Booker *et al.* [23] describe a general framework for managing surrogate objective functions to facilitate the optimization of expensive computer simulations based on the ideas of [39]. The *surrogate management framework* uses a sequence of surrogate functions to identify promising regions in which to use successively better surrogates, either by adopting models with greater physical fidelity or by constructing approximations from a greater concentration of design sites [23]. This framework uses pattern search as the optimization method, but with convergence analysis presented in [51, 52, 82], that allows great flexibility in the search methods employed to find the next iterate. For example, an extensive search on the current surrogate can be performed to select new points at which to evaluate the objective function.

Given the optimization problem (1.1), there are several assumptions defined in [23, 39]. The first assumption is that there exists a family of approximation models $\mathcal{M} = \{M^\alpha : \alpha \in \mathcal{A}\}$, where α is the index of the possible models in the set \mathcal{A} . The second assumption is that $f(x)$ is computationally expensive, but there are methods for computing or estimating $M^\alpha(x) \approx f(x)$ cheaply, where the current $M^\alpha(x)$ represents the most accurate model of $f(x)$ obtained thus far. The final assumption is that an algorithm to recalibrate the approximation models $M^\alpha(x)$ is available. This leads to the surrogate management framework [23], shown in Figure 2.4.

As discussed in [23, 39], careful use of this algorithm can lead to an approximate model that agrees with the objective function of the optimization problem (1.1). The optimization algorithms that can be used within Figure 2.4 are discussed in Section 2.2 and Section 2.3.

SURROGATE MANAGEMENT FRAMEWORK ALGORITHM

- GIVEN a set of initial points $x_i \in \mathbb{R}^n$, $i = 1, 2, \dots, n$ and an initial approximation model M^α , where $f(x_i) = M^\alpha(x_i)$
- For $k = 0, 1, \dots$, do the following:

If convergent, then exit; otherwise, continue.

SURROGATE OPTIMIZATION: Apply an optimization algorithm to the surrogate to construct a trial set $T_k = \{x_j\}_{j=1}^m$ of “good” points from $M^\alpha(x)$. If $T_k \neq \emptyset$, go to evaluate step, else go to the objective function optimization algorithm.

EVALUATE: Evaluate f on the points in T_k until an x_{k+1} is found that minimizes f on T_k or until all points in T_k have been evaluated.

If such an x_{k+1} is not found, apply one iteration of an optimization algorithm to f to try to find an x_{k+1} for which $f(x_{k+1}) < f(x_k)$.

RECALIBRATION: Recalibrate $M^\alpha(x)$ with the new values of $f(x)$ computed at points in T_k or by the optimization algorithm, return to the surrogate optimization algorithm with the updated M^α , and increment k .

Figure 2.4 Surrogate Management Framework Algorithm

2.2 Generalized Pattern Search

Generalized Pattern Search (GPS) is a class of direct search methods for non-linear optimization problems. The term “*direct search*” was first introduced in the 1960s by Hooke and Jeeves [49] as a method of making direct comparisons of objective function values without using derivative information. In 1997, Torczon [82] defined and analyzed the derivative-free class of pattern search algorithms for unconstrained optimization problems with continuously differentiable objective functions. Of importance in [82] was the result showing that a subsequence of pattern search iterates $\{x_k\} \in \mathbb{R}^n$ converges to a first order stationary point x^* ; *i.e.*, $\nabla f(x^*) = 0$.

The connection between pattern search and the positive basis theory of Davis [37] was introduced by Lewis and Torczon [51]. Pattern search was subsequently extended by Lewis and Torczon to problems with bound constraints [52] and a finite number of linear constraints [53]. In these cases, convergence theory requires the

directions used by the algorithm to be chosen at each iteration, so that they conform to the boundary of any nearby constraint.

Audet and Dennis [18], in introducing a slightly generalized version called generalized pattern search (GPS), added a hierarchy of convergence results for unconstrained and linearly constrained problems, including a new theory based on the nonsmooth calculus of Clarke [33]. Abramson [4] studied second-order behavior of GPS and showed that, under certain algorithmic choices, strict local maximizers and an entire class of saddle points can be eliminated from convergence consideration. Audet and Dennis [19] extend their approach to handle nonlinear constraints by adding a filter method [42] for GPS that accepts new iterates if improvement in either the objective function or an aggregate constraint violation function is found. Alternatively, Lewis and Torczon [55] handle nonlinear constraints by solving a sequence of bound constrained augmented Lagrangian subproblems [34].

As described by Audet and Dennis [18], the GPS algorithm consists of two distinct steps for evaluating and generating a sequence of iterates with nonincreasing function values. At each iteration, an optional SEARCH and a local POLL step are executed, in which the objective function is evaluated at a finite number of points that lie on a mesh with the goal of finding an *improved mesh point*. The mesh is defined by the set of *positive spanning* directions $D \subset \mathbb{R}^n$, where any vector in \mathbb{R}^n can be expressed as a nonnegative linear combination of these directions [37]. The use of positive spanning directions ensures the existence of at least one feasible descent direction, provided that f is differentiable and x_k is not already a stationary point. The set D (which is also represented here by a matrix whose columns are its elements) must be constructed as the product,

$$D = GZ, \tag{2.47}$$

where $G \in \mathbb{R}^{n \times n}$ is a fixed nonsingular generating matrix and $Z \in \mathbb{Z}^{n \times |D|}$ is a full rank integer matrix. Thus any direction $d_j \in D$ is represented by

$$d_j = Gz_j,$$

where z_j is a column of Z . The mesh at iteration k is centered around the current iterate x_k and can be defined (see [18]) by

$$M_k = \{x_k + \Delta_k^m Dz : z \in \mathbb{Z}_+^{|D|}\}, \quad (2.48)$$

where Δ_k^m is the mesh size parameter that controls the fineness of the mesh.

The SEARCH step does not contribute to the convergence theory. It is simply an evaluation of a finite number of mesh points that may be generated using a variety of methods, and with the goal of improving the efficiency and performance of the algorithm. Examples of these methods may include applying a heuristic, such as a genetic algorithm or randomly selecting a set of space-filling points using a Latin hypercube search. As described in Sections 2.1 and 2.1.4, when the objective function is computationally expensive to evaluate, the SEARCH step can be used to optimize a less expensive surrogate function on the mesh [13, 23, 39, 41, 77, 78].

If the SEARCH step fails to return an improved mesh point, the POLL step is invoked. The POLL step is necessary for the proof of convergence. This step consists of evaluating points that are adjacent to the current iterate x_k with respect to the positive spanning directions. This set of points is called the *poll set* and is defined by

$$P_k(x_k) = \{x_k + \Delta_k^m d : d \in D_k \subseteq D\}, \quad (2.49)$$

where D_k is a positive spanning set that is composed from the columns of D . The points in $P_k(x_k)$ are evaluated until an improved mesh point is found or until all

points of $P_k(x_k)$ have been evaluated. If the SEARCH or POLL step is successful then the iteration ends immediately with the improved point becoming the new iterate. The mesh size is retained or coarsened according to the rule,

$$\Delta_{k+1}^m = \tau^{w_k^+} \Delta_k^m, \quad (2.50)$$

where $\tau \in (1, \infty) \cap \mathbb{Q}$ remains constant over all iterations and $w_k^+ \in [0, w^+] \cap \mathbb{Z}$ is an integer bounded above by $w^+ \geq 0$. If neither step finds an improved mesh point, then x_k is said to be a *mesh local optimizer*, and the mesh size is refined or tightened according to the rule,

$$\Delta_{k+1}^m = \tau^{w_k^-} \Delta_k^m, \quad (2.51)$$

where $w_k^- \in [w^-, -1] \cap \mathbb{Z}$ and $w^- \leq -1$.

The GPS algorithm is summarized in Figure 2.5 [4, 18]. A pictorial representation of a single GPS iteration is given in Figure 2.6 [9]. The iteration begins with the evaluation of a feasible point x_0 on the mesh, followed by some type of SEARCH, which may include the formation and optimization of a surrogate. Since the surrogate optimizer does not produce improvement in the objective function value, the POLL step is invoked, and since no improvement is found, the mesh is refined and a new iteration begins from the current solution.

To handle the linear constraints in (1.1), GPS is applied using an extreme “barrier” approach [8, 20]. This algorithm is not applied to f , but to the barrier objective function $f_\Omega = f + \psi_\Omega$, where ψ_Ω is the indicator function for Ω . If an evaluated point x does not belong to Ω , $\psi_\Omega(x) = \infty$, implying that $f_\Omega(x) = \infty$, and f is not evaluated. If $x \in \Omega$, $\psi_\Omega = 0$ and f is evaluated [8]. This becomes very important in many engineering problems where f is expensive to evaluate.

A BASIC GPS ALGORITHM

- **INITIALIZATION:** Let $x_0 \in \Omega$ be such that $f_\Omega(x_0)$ is finite. Let D be a positive spanning set, and let M_0 be the initial mesh defined by an initial mesh size parameter Δ_0^m and D according to (2.48).
- **SEARCH AND POLL:** Perform the SEARCH and possibly the POLL steps (or only part of them) until an improved mesh point x_{k+1} is found on the mesh M_k .
SEARCH STEP: Evaluate f_Ω on a finite subset of trial points on the mesh M_k using some user defined strategy seeking an improved mesh point.
POLL STEP: If the SEARCH step was unsuccessful or not performed, evaluate f_Ω on the poll set $P_k(x_k)$ until an improved mesh point is found or all points in $P_k(x_k)$ have been evaluated.
- **PARAMETER UPDATE:** If SEARCH or POLL finds an improved mesh point, Update x_{k+1} and set $\Delta_{k+1}^m \geq \Delta_k^m$ according to (2.50) and go to SEARCH AND POLL steps
 Else, set $x_{k+1} = x_k$ and $\Delta_{k+1}^m < \Delta_k^m$ according to (2.51) and go to SEARCH AND POLL steps

Figure 2.5 Basic GPS Algorithm

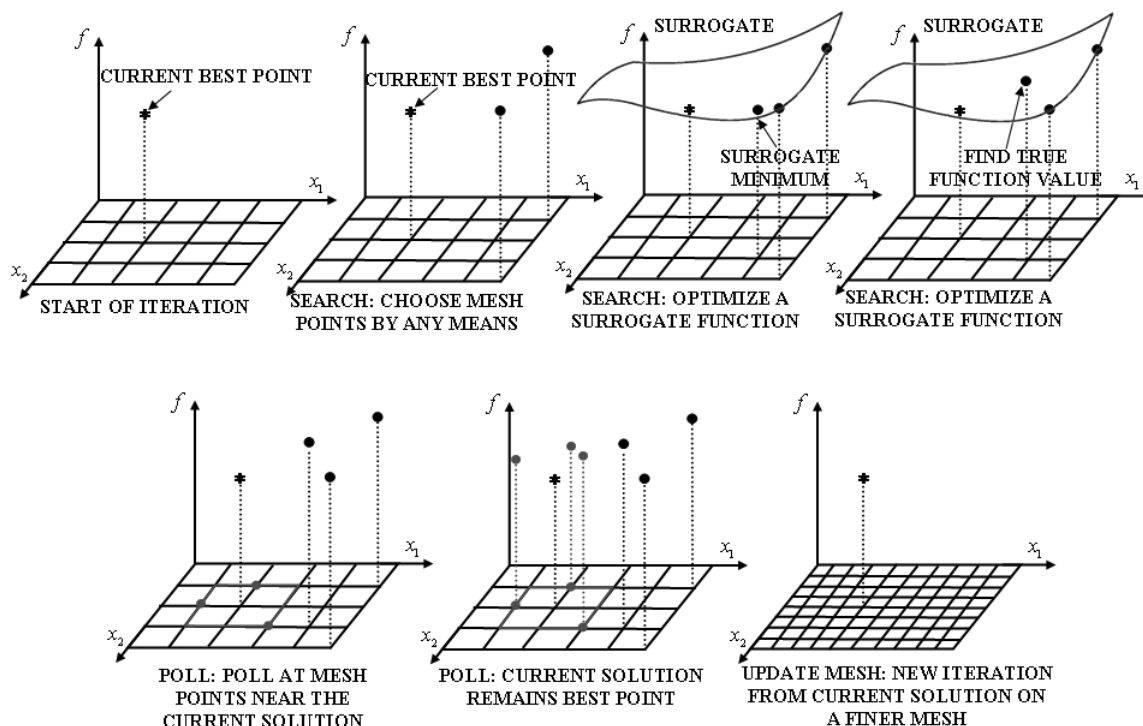


Figure 2.6 Basic GPS Algorithm-Pictorial Representation

2.2.1 Mixed Variable Pattern Search

Many optimization problems contain both discrete and continuous variables, and some discrete variables may not be integer-valued. They can take on categorical values such as type of material, color, shape, etc. Thus, standard mixed integer approaches using continuous relaxations with branch and bound to solve these problems are not possible. Audet and Dennis [17] extended GPS to mixed variable problems with bound constraints by including user-specified discrete neighborhoods in the definition of the mesh, where the objective function f is assumed to be continuously differentiable for fixed discrete variable values. The resulting Mixed Variable Pattern Search (MVPS) algorithm [17] was successfully applied to a thermal insulation system design problem in [50]. Abramson *et al.* extended the results of [17] to linear [1] and nonlinear [7] constraints, again making use of the Clarke calculus [33], and the latter being augmented with a filter [19] to handle the nonlinear constraints. Abramson [3] successfully applied the resulting algorithm to the design of a load-bearing thermal insulation system, which was a modification of the problem in [50]. These algorithms were also used to quantitatively reconstruct objects from x-ray radiograph data [12, 68].

In order to describe the mesh and poll sets, the discrete neighbors $\mathcal{N}(x_k)$ of x_k must be defined. Audet and Dennis [17] and Abramson *et al.* [7] describe $\mathcal{N}(x_k)$ as a necessity for the extension of the algorithm, and local optimality is proved with respect to this set. The discrete neighbor set $\mathcal{N}(x_k)$ is finite and consists of the current iterate x_k and other points that at least differ in the discrete values. A common choice of a discrete neighbor set when the discrete variables are integer-valued is

$$\mathcal{N}(x_k) = \{y \in \Omega : y^c = x_k^c, \|y^d - x_k^d\|_1 \leq 1\}; \quad (2.52)$$

i.e., each discrete neighbor of x_k is constructed by holding the continuous variables constant and changing one discrete variable at a time by a single unit. However, as stated in [7, 17], when the discrete variables are categorical, the discrete neighbor set may not be well-defined since no underlying topology is assumed. Thus, the local neighborhood must be defined by the user, based on *a priori* knowledge of the physical problem.

The definition of local optimality is extended to account for variation of both the continuous and discrete variables [17]. It is defined as follows [7, 17]. A point $x = (x^c, x^d) \in \Omega$ is said to be a local minimizer of f on Ω with respect to the set of neighbors $\mathcal{N}(x)$ if there exists an $\varepsilon > 0$ such that $f(x) \leq f(v)$ for all v in the set

$$\Omega \cap \bigcup_{y \in \mathcal{N}(x)} (B(y^c, \varepsilon) \times \{y^d\}), \quad (2.53)$$

where $B(y^c, \varepsilon)$ is the open ball of radius ε around y^c .

In the mixed variable case, the positive spanning directions are defined slightly differently than in Section 2.2. For each combination $i = 1, 2, \dots, i_{max}$ of values the discrete variables can take on, the positive spanning directions $D^i \subset \mathbb{R}^{n^c}$ are constructed by:

$$D^i = G_i Z_i, \quad (2.54)$$

where $G_i \in \mathbb{R}^{n^c \times n^c}$ is a fixed nonsingular generating matrix and $Z_i \in \mathbb{Z}^{n^c \times |D^i|}$ is full rank integer matrix [7]. The mesh M_k at iteration k centered around the current iterate x_k as the product of Ω^d with the union of a finite number of lattices in Ω^c , each of which is centered at the continuous part of the current iterate: *i.e.*,

$$M_k = \bigcup_{i=1}^{i_{max}} M_k^i \times \Omega^d \quad (2.55)$$

with $M_k^i = \bigcup_{x \in S_k} \{x_k^c + \Delta_k^m D^i z \in \Omega^c : z \in \mathbb{Z}_+^{|D^i|}\} \subset \mathbb{R}^{n^c}$,

where S_k are all the previously evaluated trial points at iteration k [7]. The poll set in MVPS is defined with respect to the continuous variables, the discrete neighbor points, and the set of points generated by an EXTENDED POLL step. At the k th iteration, let $D_k^i \subseteq D^i$ represent the set of positive spanning directions for the poll set corresponding to the i th set of discrete variable values and define $D_k = \bigcup_{i=1}^{i_{max}} D_k^i$. The poll set centered at the current iterate x_k , as defined by [7], is:

$$P_k(x_k) = \{x_k\} \cup \{x_k + \Delta_k^m(d, 0) \in \Omega : d \in D_k^i\} \subset M_k, \quad (2.56)$$

where $(d, 0)$ represents the partitioning into continuous and discrete variables respectively, with 0 meaning that the discrete variables remain unchanged; *i.e.*, $x_k + \Delta_k^m(d, 0) = (x_k^c + \Delta_k^m d, x_k^d)$.

If the POLL step fails to return an improved mesh point from the poll set with respect to the continuous variables or the discrete neighbors, then MVPS performs an EXTENDED POLL step. Given a neighbor $y \in \mathcal{N}(x_k)$ satisfies

$$f(x_k) \leq f(y) < f(x_k) + \xi_k, \quad (2.57)$$

where $\xi_k \geq \xi$ is a user-specified *extended poll trigger*, with ξ a fixed positive scalar, then a finite number of POLL steps are evaluated around the points satisfying (2.57) [7]. The value for ξ_k is usually chosen as some percentage of the objective function value, bounded away from zero, such as $\xi_k = \max\{\xi, 0.05|f(x_k)|\}$. Higher choices of ξ_k lead to better solutions but at the cost of more function evaluations, while a lower value may yield a poorer solution but requires fewer function evaluations [7, 17]. The set of points evaluated by the EXTENDED POLL step at iteration k can be expressed as

$$\mathcal{X}_k(\xi_k) = \bigcup_{y \in \mathcal{N}_k^{\xi_k}} \bigcup_{j=1}^{J_k} P_k(y_k^j), \quad (2.58)$$

where $\mathcal{N}_k^{\xi_k} = \{y \in \mathcal{N}(x_k) : f(x_k) \leq f(y) < f(x_k) + \xi_k\}$. The traditional assumption that all iterates lie in a compact set ensures that J_k (and thus, the number of points in $\mathcal{X}_k(\xi_k)$) is finite.

Given the addition of the EXTENDED POLL step, the mesh size can be retained or coarsened according to (2.50), and the mesh size is refined or tightened according to (2.51). The MVPS algorithm can be seen in Figure 2.7 [1, 7, 17], followed by a pictorial representation of the algorithm in Figure 2.8 [9]. The only difference in Figure 2.6 and the Mixed Variable GPS algorithm lies in the POLL step. The evaluation of the POLL set, discrete neighbor set, and possibly the EXTENDED POLL set of points is represented in the figure.

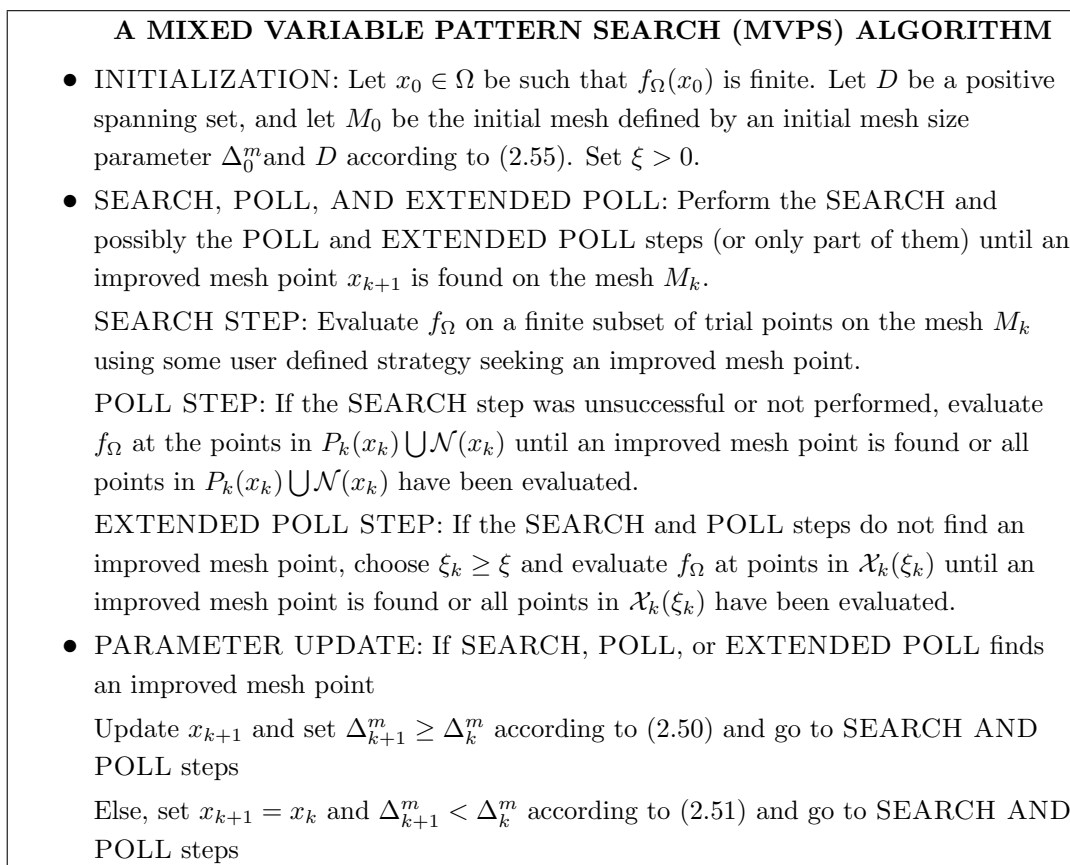


Figure 2.7 Mixed Variable GPS Algorithm

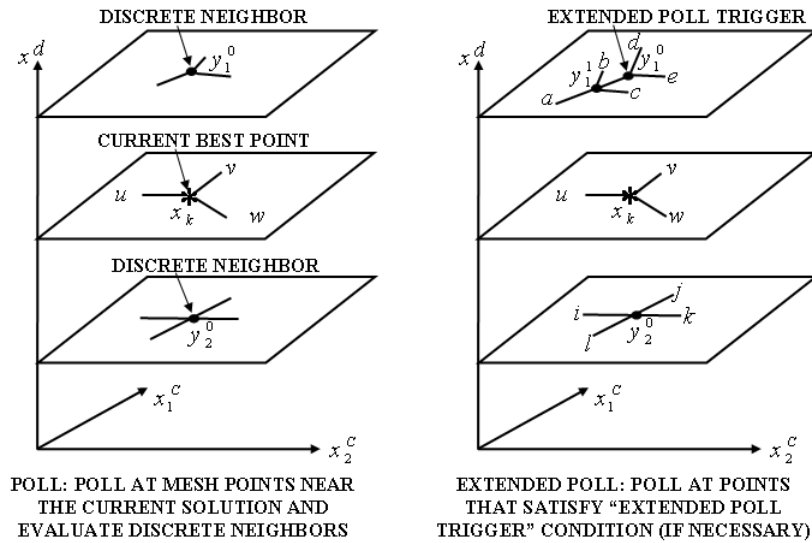


Figure 2.8 Mixed Variable GPS Algorithm-POLL Step Illustration

2.3 Mesh Adaptive Directed Search

Mesh Adaptive Direct Search (MADS) is relatively a new class of algorithms introduced by Audet and Dennis [20] as an extension of GPS to solve optimization problems with nonlinear constraints. It does not make use of filters [19] or penalty functions [55]. The MADS algorithms are organized in a similar fashion to that of the *frame-based* methods of Coope and Price [35]. Audet and Dennis [20] propose using a less general choice of *frame* (previously called a *poll set*), which is easy to implement and enables the parameter space to be searched in an asymptotically dense set of directions, thereby leading to stronger convergence theory than that of GPS [20]. The convergence analysis once again applies the Clarke nonsmooth calculus [33] to show that a subsequence of iterates of an implementable instance of MADS (in which positive spanning directions [37] are chosen in a random manner) converges almost surely to a first-order stationary point. Abramson and Audet [5] show that under reasonable additional assumptions, a subsequence of MADS iterates converges to a point that satisfies second-order optimality conditions.

MADS uses the same two SEARCH and POLL steps described in Section 2.2. The SEARCH step is the same as in GPS, where the points are restricted to lie on the mesh M_k , defined by (2.48). This is an idea that makes MADS less general than the frame methods of Coope and Price [35]. If the SEARCH step fails to find an improved mesh point, the POLL step is invoked. This is where the difference in the MADS and the GPS algorithms lies. The MADS algorithm introduces a new parameter, known as the *poll size parameter* $\Delta_k^p \in \mathbb{R}_+$, which dictates the maximum distance a trial point generated by the POLL step can be from the current incumbent solution x_k [20]. In GPS, $\Delta_k^p = \Delta_k^m$. In each MADS iteration, the mesh size parameter is updated as in GPS and the poll size parameter is updated such that $\Delta_k^m \leq \Delta_k^p$ for all k and

$$\lim_{k \in K} \Delta_k^m = 0 \Leftrightarrow \lim_{k \in K} \Delta_k^p = 0 \text{ for any infinite subset of indices } K. \quad (2.59)$$

The points evaluated during the POLL step are based on the construction of a MADS frame. The frame is constructed from the current best solution x_k (or *frame center*), poll size parameter (Δ_k^p), mesh size parameter (Δ_k^m), and a set of positive spanning directions D_k . This set is different than the POLL set in GPS, in that D_k is not necessarily composed of the columns of D (i.e. $D_k \not\subseteq D$). The MADS *frame* or POLL set is defined as

$$P_k(x_k) = \{x_k + \Delta_k^m d : d \in D_k\}, \quad (2.60)$$

where each $d \in D_k$ must satisfy the following three properties [20]:

- $d \neq 0$ can be written as a nonnegative integer combination of the directions in D : $d = Du$ for some vector $u \in \mathbb{Z}_+^{|D|}$ that may depend on iteration number k ,

- The distance from the frame center x_k to a frame point $x_k + \Delta_k^m d \in P_k(x_k)$ is bounded by a constant times the poll size parameter:

$$\Delta_k^m \|d\| \leq \Delta_k^p \max\{\|d'\| : d' \in D\},$$
- Similar to Coope and Price [35], the limits of the normalized sets D_k are positive spanning sets.

The mesh size parameter is updated at each iteration of the algorithm according to the same methodology presented in Section 2.2. It is coarsened according to (2.50) and refined according to (2.51).

The MADS algorithm is summarized in Figure 2.9, which is similar to the GPS algorithm in Figure 2.5 with differences in the POLL step and in the addition of the poll size parameter [5, 20]. To solve (1.1), MADS is applied using the extreme “barrier” approach described in Section 2.2. The barrier approach is also applied to nonlinear constraints in MADS.

Figures 2.10 and 2.11, which are taken from [20], illustrate the difference in the frames of GPS and MADS. The GPS frames in Figure 2.10 are generated such that $\Delta_k^m = \Delta_k^p$; therefore, the number of positive spanning sets composed of the columns of D is finite over all iterations. The MADS frames in Figure 2.11 are generated using $\Delta_k^p = n\sqrt{\Delta_k^m}$, where n is the dimensionality of the parameter space. The advantage of MADS over GPS is that the mesh size parameter Δ_k^m typically decreases to zero at a faster rate than the poll size parameter Δ_k^p , which allows the set of directions in D_k used to define the MADS frame (2.60) to be chosen from increasingly larger sets (as a limit point is approached). Audet and Dennis [20] show that if this set is dense in the limit, convergence to a stationary point in the nonsmooth case can be ensured. They also provide an implementable instance, in which directions are chosen randomly and a dense set of directions is achieved with probability one [20].

A BASIC MADS ALGORITHM

- **INITIALIZATION:** Let $x_0 \in \Omega$ be such that $f_\Omega(x_0)$ is finite. Let D be a positive spanning set, and let M_0 be the initial mesh such that the initial mesh size parameter $\Delta_0^m \leq \Delta_0^p$ according to (2.48, 2.59).
- **SEARCH AND POLL:** Perform the SEARCH and possibly the POLL steps (or only part of them) until an improved mesh point x_{k+1} is found on the mesh M_k .
SEARCH STEP: Evaluate f_Ω on a finite subset of trial points on the mesh M_k using some user defined strategy seeking an improved mesh point.
POLL STEP: If the SEARCH step was unsuccessful or not performed, evaluate f_Ω on the frame $P_k(x_k)$ until an improved mesh point is found or all points in $P_k(x_k)$ have been evaluated.
- **PARAMETER UPDATE:** If SEARCH or POLL finds an improved mesh point, Update x_{k+1} , set $\Delta_{k+1}^m \geq \Delta_k^m$ according to (2.50), and Δ_{k+1}^p according to (2.59) and go to SEARCH AND POLL steps
 Else, set $x_{k+1} = x_k$, $\Delta_{k+1}^m < \Delta_k^m$ according to (2.51), and Δ_{k+1}^p according to (2.59) and go to SEARCH AND POLL steps

Figure 2.9 Basic MADS Algorithm

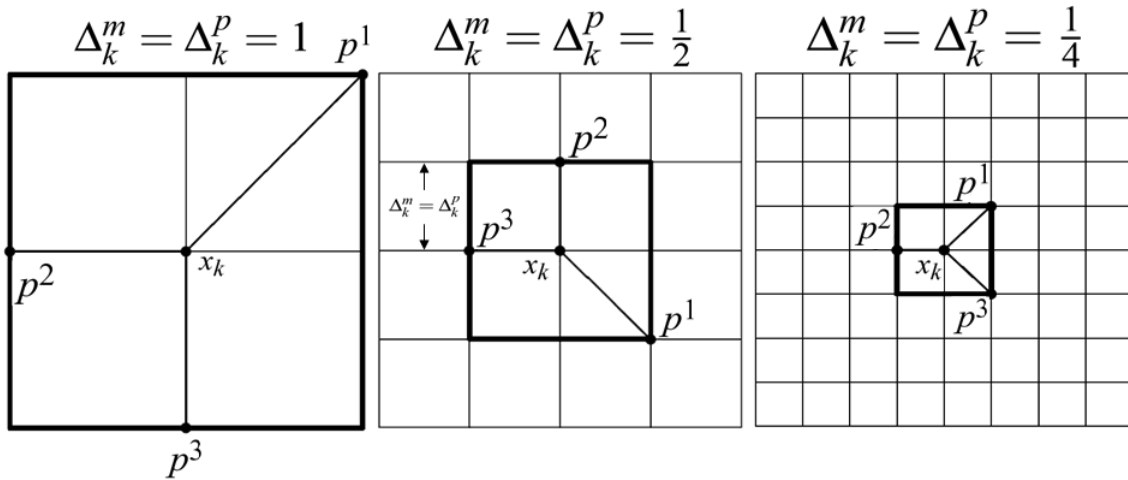


Figure 2.10 A GPS FRAME

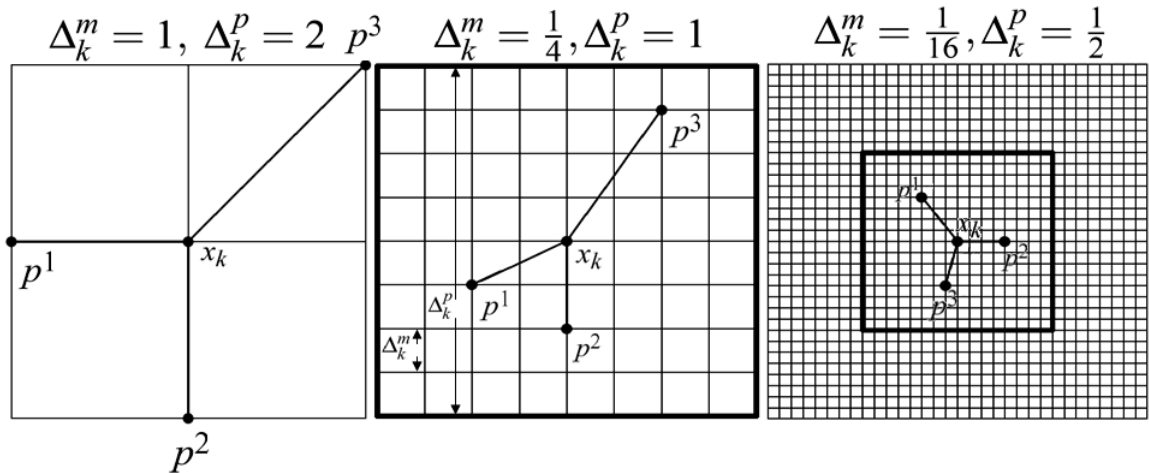


Figure 2.11 A MADS FRAME

2.3.1 Mixed Variable Mesh Adaptive Direct Search

Mixed Variable Mesh Adaptive Direct Search (MVMADS) was introduced as a generalization of MADS to mixed variable optimization problems by Abramson *et al.* [8]. Just as seen in [1, 7, 17], local optimality conditions are defined in terms of the local neighborhoods discussed in Section 2.2.1. Each iteration of MVMADS is similar to that of MVGPS with the difference being in the POLL and EXTENDED POLL steps.

The mesh M_k is consistent with the one defined in [7] and described by (2.55). The POLL step is based on applying the MADS frame defined in [20] and (2.60) to the POLL set (2.56) and the discrete neighborhood (2.52) defined by Audet and Dennis [17] and Abramson *et al.* [7]. Due to the categorical variables, there are additional evaluations using the EXTENDED POLL step [7, 17], where the set of points to be evaluated is determined by applying a MADS frame to the set of points described in (2.58).

The extended poll trigger is chosen consistent with the discussion in Section 2.2.1. The rules for coarsening and refining the mesh size are the same as those given in (2.50) and (2.51), respectively. The poll size parameter must be updated

in the same manner described in Section 2.3 while satisfying equation (2.59). The MVMADS algorithm of Abramson *et al.* [8] is shown in Figure 2.12.

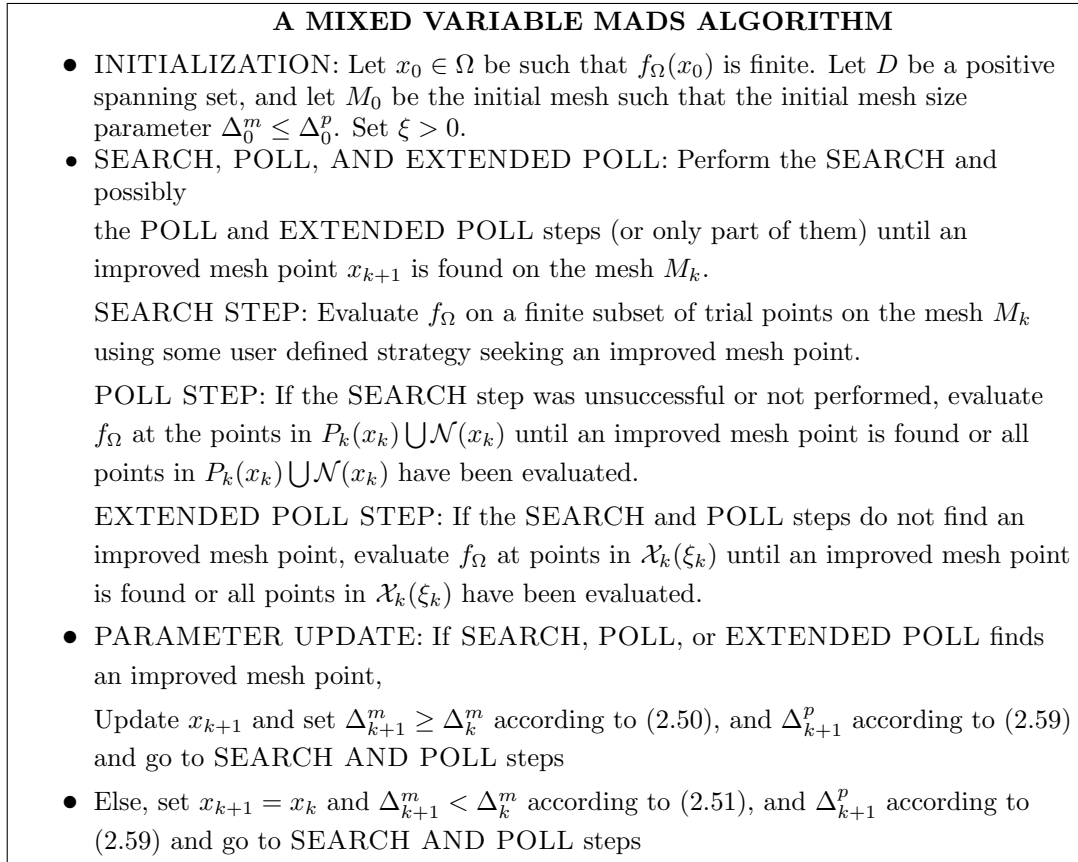


Figure 2.12 Mixed Variable MADS Algorithm

2.4 Surrogates Based on CPU Time

Optimization problems that make use of CPU subprocess times to control how the next set of parameter values to test are chosen were first studied by Magallanez [60]. His work consisted of reducing the computation time required to recover model parameters for problems that exhibit a strong correlation between objective function values and the computational time associated with obtaining them (in this case, the numerical image registration process). The use of CPU subprocess times to monitor

and control computing resources was described in Section 1.2 and displayed as dashed lines in Figure 1.1, under the “*cputime2*” label.

Magallanez [60] sought to minimize a computationally expensive black box function similar to (1.1) but only with continuous variables; *i.e.* where $\Omega = \{x \in \mathbb{R}^n : l \leq x \leq u\}$ and $l, u \in (\mathbb{R} \cup \{\pm\infty\})^n$ for $l < u$. The problems he studied exhibited the property that the computational time required to evaluate f , at a point x decreases as x approaches the solution [60]. To exploit this property, input and output arguments were added to the function; namely,

$$[z, t] = f(x, t_k^{cut}), \quad (2.61)$$

where $x \in \Omega$ is the trial point, t_k^{cut} is a computational time cut-off threshold, z is the function value at x , and t is the time needed to compute z .

The idea used in [60] was that if the computational time exceeded the t_k^{cut} value, then the evaluation of f should stop because a lower objective function value will probably not occur in the increased amount of computational time, due to the CPU-time correlation property. Magallanez [60] used a method that allowed the t_k^{cut} parameter to change according to the computational time associated with the best incumbent solution found thus far.

The SEARCH step used in [60] was to solve, at little computational cost, four individually formed surrogate problems to find a point at which to evaluate f . The four surrogate problems consisted of:

$$\min_{x \in \Omega} F(x) \quad (2.62)$$

$$\min_{x \in \Omega} F(x), \quad \text{s.t. } T(x) \leq t_k^{cut} + \varepsilon \quad (2.63)$$

$$\min_{x \in \Omega} T(x) \quad (2.64)$$

$$\min_{x \in \Omega} T(x), \quad \text{s.t. } F(x) \leq \hat{z}_s, \quad (2.65)$$

where ε is added to allow for variability in computational time.

The surrogates $F(x)$ and $T(x)$ were constructed based on a set of initial model parameters values and their associated objective function and computational time values, respectively. These surrogates are updated each time a new set of parameters is evaluated by the objective function. The surrogates with constraints, (2.63) and (2.65), ensure a decline in both functional and time values given a set of input parameters. The idea is similar to that discussed in Section 2.1.4, where an improved solution may be found quickly through the use of a surrogate without spending increased amounts of time on the computationally expensive objective function.

The surrogates (2.62)–(2.65) were constructed using DACE (see Section 2.1.2) with a second-order regression polynomial and a *Gaussian* correlation model (see (2.31) and (2.33), respectively). The choice of polynomial order and correlation model was based on [63] and [57], where a second-order polynomial showed improved accuracy as a predictor and a *Gaussian* correlation model showed behavior similar to the desired function as the number of known points increases. The set of initial points was determined using an *inscribed* Central Composite Design (CCD) that ensures all initial points are within the feasible region. (A more detailed description of CCD will be given in Chapter 3.)

To handle the possible ill-conditioning (see Booker [25]) of $R(\theta, x_i, x_j), i, j = 1, \dots, k$, when solving (2.29) and (2.30) as points begin to cluster during the optimization process, Magallanez [60] periodically reviews the condition number of the Cholesky factor C of R (where $R = CC^T$ is the Cholesky decomposition),

$$\kappa(R) \leq \kappa(C)^2 = \frac{\lambda_{max}^2(C)}{\lambda_{min}^2(C)}$$

and where λ_{max} and λ_{min} are the largest and smallest eigenvalues of C , respectively. If $\kappa(C)$ was too large, the surrogate would not be formed and the SEARCH step would be skipped, allowing only the POLL step to execute [60]. The algorithm,

called MADS-TIME, which executes either GPS or MADS, is fully detailed in [60].

Magallanez [60] ran the MADS-TIME algorithm on two test problems, which will be described in Chapter 4: a *Lid-Driven Cavity* and a *Barrier Flow* problem. The results of the implementation of the surrogates (2.62)–(2.65) and the analysis of these results for the *Lid-Driven Cavity* problem are given in [60]. The results of the *Barrier Flow* problem yielded much different results. GPS and MADS were both used as the optimization method, but neither was found effective at recovering the optimal parameters. An additional run was performed to investigate and illustrate the reasons why it was thought to be impractical to find an optimal solution to this problem. More details of the results and investigation are discussed in [60].

2.5 Conclusion

This chapter reviewed the relevant literature and provided an introduction to surrogates and relevant direct search methods. The combination of these two ideas as applied to solving continuous and mixed variable optimization problems was also discussed. In the next chapter, a methodology is developed using the combination of these ideas to numerically solve the mixed variable optimization problem (1.1) more efficiently.

3. Methodology

This chapter outlines the approach for solving the optimization problem presented in Chapter 1, with the goal of recovering the model parameters at minimum computational expense for this type of application problems. A description to efficiently solve this problem through the use of multiple surrogates and mixed variable direct search methods within an optimization framework are developed within this chapter.

3.1 *Mixed Variable Optimization Problem and Notation*

The mixed variable optimization problem is defined in (1.1) of Section 1 over the domain Ω (feasible region), which is the union of continuous variable spaces Ω^c across possible discrete variable spaces Ω^d ; *i.e.*,

$$\Omega = \bigcup_{x_d \in \Omega^d} (\Omega^c(x^d) \times \{x^d\}),$$

where Ω^c and Ω^d are also defined in Section 1. The goal is to find a set of parameters that minimizes (1.1) as it pertains to an application.

As mentioned in Sections 1.1 and 1.2, the objective is to recover certain model parameters at minimal simulation fidelity of the computational parameters while reducing the amount of time spent on unnecessary computations. By extending [60] to mixed variables, the optimization framework can make use of the computational parameters to mitigate the expense of an evaluation of (1.1). This implies that the computing resources required for an objective function evaluation can be more closely monitored using additional CPU time information, as was illustrated by the dashed lines (labeled “cputime1” and “cputime2”) in Figure 1.1. As in [60], the first step was to add this information as an input and output to the objective function;

namely,

$$[z, t] = f(x, t_k^{cut}), \quad (3.1)$$

where x is a trial point, z is the function value at x , t is the computational time of computing z , and t_k^{cut} is the computational time threshold allowed for computing z at the point x . If the computational time exceeds the t_k^{cut} value, the current objective function evaluation is stopped. This is done to control the amount of time spent on unprofitable computations, and because a decreased objective value is unlikely to occur at the current values of the model and computational parameter choices in an increased amount of computation time. In [60], the t_k^{cut} parameter was updated based only on the “cputime2”, in Figure 1.1, associated with the minimal objective function value. However, the inclusion of computational parameters in (1.1) requires the method used for updating t_k^{cut} to be altered according to both the computational and model parameters associated with the minimal objective function value; *i.e.*, given a current minimal function value z_k with associated computational time t_k at iteration k , set

$$t_k^{cut} = \alpha^{cut} t_k, \quad (3.2)$$

where t_k is the CPU time required to compute z_k and $\alpha^{cut} > 1$. For the class of mixed variable problems targeted here and defined by the CFD simulation and image registration times associated with the current minimal function value, t_k is given by

$$t_k = t_k^{sim} + t_k^{image}. \quad (3.3)$$

The parameter $\alpha^{cut} > 1$ is a user-specified multiplier that allows for variation in computational time of the processes associated with an objective function evaluation. This allows the t_k^{cut} variable to be set such that function evaluations can occur that possibly have slightly increased computational times.

3.2 Customized Search: Optimization using Multiple Surrogates

The optional SEARCH step of the direct search methods described in Sections 2.2 and 2.3 can make use of the additional computational time data. Given a set of evaluated points X_k and associated objective function values and computational times, surrogate functions $F(x)$ and $T(x)$ are constructed based on function values and their computational times, respectively. Minimizing $F(x)$ is performed with little computational expense at each SEARCH step, and $T(x)$ is used to determine the order in which trial points found during the optimization of $F(x)$ are evaluated.

The surrogate optimization of $F(x)$ is solved by applying the same direct search method (using an extreme barrier approach) as described in Sections 2.2 and 2.3. However, the discrete variables x^d are treated differently in the surrogate optimization problem. Both discrete and continuous parameter values of the points in X_k used in the construction of $F(x)$ are treated as continuous variables during the surrogate optimization process. Since this may yield infeasible SEARCH points with respect to the discrete variables, all trial points must be monitored and, if required, undergo a transformation according to the following rules:

- Let S_k represent the finite set of n_S trial points returned by the SEARCH step while optimizing the surrogate problem on $F(x)$. (The number n_S is user-specified.)
- For each $y \in S_k$ such that $y^d \notin \Omega^d$, replace y with $\{(y^c, \lfloor y^d \rfloor), (y^c, \lceil y^d \rceil)\}$, where $\lfloor y^d \rfloor = \max\{z \in \mathbb{Z} : z \leq y^d\}$ and $\lceil y^d \rceil = \min\{z \in \mathbb{Z} : z \geq y^d\}$.

Depending on the number of required transformations, each time the SEARCH step is invoked, the number of trial points returned for objective function evaluation ranges from m to $2m$ points.

The surrogate function $T(x)$ is not optimized during the SEARCH step. Instead, $T(x)$ is used to order the trial points returned from the surrogate optimization problem on $F(x)$. This is done with the expectation that trial points with lower

computational times result in a decreased objective function value, thus not requiring all the trial points to be evaluated and reducing the time spent on unprofitable computations. The trial points are ordered in the following manner:

- Let the n_S to $2n_S$ trial points returned from the optimization of $F(x)$ be inputs into $T(x)$, where the responses, $\{T(y) : y \in S_k\}$, are the predicted computational times associated with the predicted functional values of $\{F(y) : y \in S_k\}$.
- The n_S best trial points are selected and evaluated in the objective function based on the increasing predicted computational time.

The SEARCH points are evaluated by the objective function until a new incumbent solution is found or until all points have been evaluated. If a decrease in f_Ω is not found, the SEARCH is considered unsuccessful, and the POLL (and possibly the EXTENDED POLL) step is performed according to the methods described in Sections 2.2.1 and 2.3.1. The collection of trial points evaluated throughout these steps and their associated objective function response and computational time are added as design sites and responses, respectively. The surrogate functions are reconstructed (recalibrated) according to Section 2.1.4. If an objective function value of ∞ is returned, due to a possible divergent image registration solution (see Section 1.2.3) or infeasible points not in Ω , the parameter values and responses are not used in the surrogate function recalibration.

3.3 Composition of Surrogate Functions

The surrogate functions, $F(x)$ and $T(x)$, described in Section 3.2 were constructed using either DACE surrogates or RBFs (see Sections 2.1.2 and 2.1.3, respectively). The following subsections discuss the methods for generating the initial points and the order of the polynomial used for the regression model in both surro-

gate types. The DACE correlation model and specific RBF types used to predict the distances between known and unknown points are also described.

3.3.1 Initial Points

Construction of the surrogates requires a set of initial points to be evaluated. As discussed in Section 2.1.2, the types of designs used can be divided into classical and modern design of experiments [45]. The goal of the design is to maximize the amount of information gained while limiting the number of sample points [74, 75]. The idea is that the initial points be “*space-filling*”, recognizing that it is very difficult to “*fill space*” in high dimensions, but enough points are sampled such that an accurate initial surrogate can be constructed [25]. Several methods from classical and modern design of experiments can be considered for determining the set of initial points, including central composite design (CCD), Latin hypercube [80, 81], and orthogonal array (OA) sampling [69].

Latin hypercube and orthogonal array samplings are commonly performed to generate a random set of initial points while accurately extracting trend information. A Latin hypercube design allows the user to tailor the number of design points, based on a limited computational budget. An example of a Latin hypercube design in \mathbb{R}^2 , with four design points, is illustrated in Figure 3.1. This design is created by dividing each of the n dimensions into m intervals of equal length, where m is the desired number of design points. This creates m^n bins for the given space. Random points within the m random bins are chosen, such that each column for each dimension is chosen only once [75]. Due to the randomness of the sampling, it is possible to achieve an ideal sampling (right), but it is also possible to achieve the poor sampling (left). Not only is the latter sampling a poor arrangement with respect to coverage of the design space, but the sample sites would have high spatial correlation, which could lead to an ill-conditioned system when the sites are being used for the surrogate construction [45].

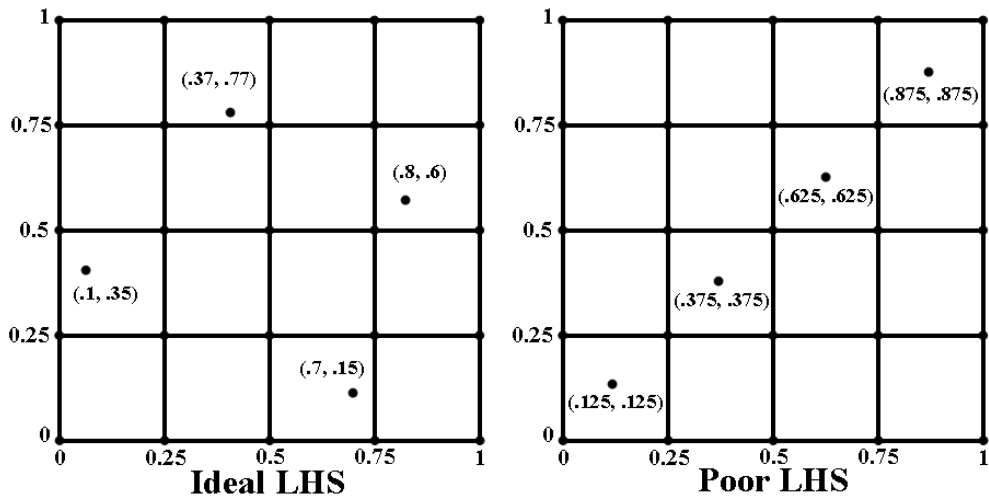


Figure 3.1 Latin Hypercube Samplings

Orthogonal array samplings are similar to Latin hypercubes. Orthogonal array samplings produce a set of samples in any t -dimensional projection on an n -dimensional design space, where t is the *strength* and $t < n$ [45]. For a strength of $t = 1$, an orthogonal array is a Latin hypercube. A disadvantage of orthogonal array sampling is that the user does not have the ability to specify the number of samples if $t \geq 2$. The generation can also be nontrivial, since several valid permutations of the orthogonal array may exist. These designs possibly need an increased number of sample points to accurately estimate coefficients of higher order terms in the surrogates. Due to the computational expense of function evaluations, classical design of experiments methods are used.

To minimize the number of samples while properly estimating higher order terms in the surrogate, the set of initial points was determined using an *inscribed* Central Composite Design (CCD). There are other types of CCDs such as *circumscribed* or *face-centered*, but they have drawbacks. The circumscribed design maintains orthogonality of sample points but includes points outside the feasible region, and the face-centered design ensures feasibility of the sample points but does not maintain orthogonality. The inscribed CCD ensures both orthogonality and fea-

sibility. This ensures that no unprofitable computations are made while evaluating the initial points to be used in the construction of the initial surrogate functions. A minimum of $2^n + 2n$ sample points are required to accurately estimate higher order terms using a circumscribed or inscribed design [63]. If the dimensionality of the problem is large, the number of sample points grows exponentially causing the evaluation of the initial points to become very expensive. In this case, Latin hypercube or orthogonal array samplings may be more efficient [45]. The test problems studied in Chapter 4 do not have this problem, since all of them have very low dimensionality.

Figure 3.2 is an illustration of the two-dimensional inscribed CCD used in this research, where the model parameters $x^c \in \mathbb{R}^2$ and the computational parameter x^d is held constant during the generation of the initial points. The center point and the axial points provide accurate estimates of quadratic regression coefficients, while the $(\pm\sqrt{2}/2, \pm\sqrt{2}/2)$ points enable the approximation of linear and interaction terms [63]. A CCD usually includes multiple replications of the center point; however, because there is no randomness in the responses and objective function evaluations are expensive, replications are omitted.

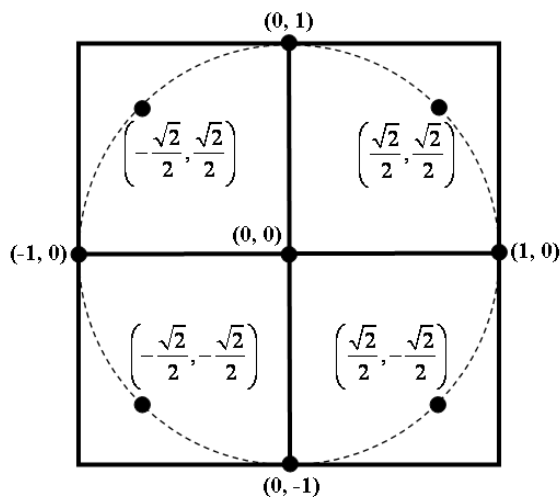


Figure 3.2 Inscribed Central Composite Design (2-D)

3.3.2 Regression Model

A second-order polynomial is a more accurate predictor of $\hat{y}(\tilde{x})$ than a polynomial of a lower order form [63]. Therefore, in order to minimize a DACE surrogate while obtaining a proper estimate of $y(x)$ for an untried point \tilde{x} , the following second-order polynomial predictor was used as the regression model,

$$\hat{y}(\tilde{x}) = \beta_0^* + \sum_{i=1}^k \beta_i^* \tilde{x}_i + \sum_{i=1}^k \beta_{ii}^* \tilde{x}_i^2 + \sum_{i=1}^k \sum_{j<i}^k \beta_{ij}^* \tilde{x}_i \tilde{x}_j + R(\theta, x, \tilde{x}) \gamma^*, \quad (3.4)$$

where β^* and γ^* come from (2.28) and are defined by (2.29)–(2.30), and the selections for $R(\theta, x, \tilde{x})$ are restricted to (2.32)–(2.33), all of which are defined in detail in Section 2.1.2. For comparison purposes a second-order polynomial was also used for the RBF surrogates; *i.e.*,

$$s(\tilde{x}) = p(\tilde{x}) + \sum_{i=1}^n \lambda_i \phi(\|\tilde{x} - x_i\|), \quad x \in \mathbb{R}^n, \quad (3.5)$$

where $p(\tilde{x})$ is a polynomial of degree two in n variables defined by (2.43), and the basis function ϕ is defined by (2.37)–(2.42) and is described in Section 2.1.3. The coefficients of the polynomial and the weights $\lambda_i = (\lambda_1, \lambda_2, \dots, \lambda_n)$ on the basis function ϕ are the solutions of the system (2.46) that define the RBF interpolant (2.36).

Initially during the optimization, it is possible that the number of design sites required to utilize a second-order regression polynomial model may not exist. Therefore, a lower-order model may be used until enough design points are available to use the specified regression model. The adaptive algorithm shown in Figures 3.4–3.5 makes use of this idea.

3.3.3 Correlation Model and Basis Function

The DACE correlation models given in (2.33), exhibit either parabolic or linear behavior near the origin. A common choice used in practice for physical phenomena is the *Gaussian* correlation model (see (2.33)), which shows similar behavior to the desired function as the number of points increases [57].

Booker [25] observed that as points are added to the initial surrogate for recalibration, the points begin clustering as the optimal objective function value is approached. This results in the matrix R in (2.25) becoming ill-conditioned, which makes it very difficult to compute R^{-1} accurately. This can result in a poor solution for the optimal θ (2.34) or prevent the calculation of β^* and γ^* in (2.29)–(2.30).

One solution to the ill-conditioning problem is to use the singular value decomposition (SVD) of R to approximate the inverse [25]. The SVD of R is given by

$$R = USV^T, \quad (3.6)$$

where U and V are orthogonal matrices (*i.e.*, $U^T U = I$, $V^T V = I$) and S is a diagonal matrix containing the singular values $s_1 \geq \dots \geq s_n \geq 0$ as its diagonal entries. It follows from (3.6) that $R^{-1} = VS^{-1}U^T$. To approximate the R^{-1} , an ϵ cut-off value is used for the singular values, and a diagonal matrix E is constructed with diagonal entries

$$e_i = \begin{cases} 1/s_i, & \text{if } s_i/s_{max} \geq \epsilon \\ 0, & \text{otherwise.} \end{cases} \quad (3.7)$$

Therefore, the approximate inverse \tilde{R}^{-1} of R is computed as

$$R^{-1} \approx \tilde{R}^{-1} = VEU^T \quad (3.8)$$

and is used to compute β^* and γ^* in (2.29)–(2.30). If R is not ill-conditioned and is positive definite, the Cholesky factor C of R can be used to compute β^* and γ^* , where $R = CC^T$ is the Cholesky decomposition.

The choice of basis functions for RBF surrogates, given by (2.37)–(2.41), have different advantages for each form. A common choice when interpolating points in two dimensions is the *Multiquadric* function, given by

$$\phi(r) = (r^2 + \gamma^2)^k, \quad k > 0, \quad k \notin \mathbb{N}. \quad (3.9)$$

Franke [43] performed a numerical study of scattered data interpolation and found that this basis function provided the most accurate interpolation surfaces of all the basis function forms for interpolation in two dimensions.

As with DACE surrogates, when points begin to cluster, the matrix $\hat{\phi}$ in (2.46) can become ill-conditioned. This can lead to inaccurate estimates of the coefficients $c = (c_1, \dots, c_m) \in \mathbb{R}^m$ of the polynomial (2.43) and the weights $\lambda = (\lambda_1, \dots, \lambda_n) \in \mathbb{R}^n$ of the basis function (2.42). Therefore, the same method using SVD (3.6)–(3.8) can be applied to the system (2.46) to help prevent ill-conditioning and compute accurate estimates of c and λ .

At every iteration, the surrogates are recalibrated and reconstructed. The condition number of R and $\hat{\phi}$ are monitored after each recalibration. If the value is too large, even after the implementation of SVD, the surrogate is not used and the SEARCH step is omitted, allowing only the POLL step to be executed.

3.4 Trust Region

The trust region approach applied here to the surrogates is based on the “*trust radius*” or “*move limit*” ideas presented in [13]. If a trust region is used, the upper and lower bounds over which the surrogate is optimized are set as a percentage α^{TR}

of the original bounds defined by Ω . These surrogate bounds can change at each iteration based on the trust region and the current incumbent solution.

The user can also specify the number of points allowed for recalibration, based on the parameter values that are within the trust region at each iteration. If the number of points is not specified, the trust region sets the surrogate bounds, but all evaluated points are used for the recalibration. The trust region is formed and surrogate bounds are updated at each iteration using the following methods:

- Let X_k be the set of all previously evaluated points at iteration k , and let $x_k = (x_k^c, x_k^d) \in X_k$ be the current best solution, with associated function value z_k .
- For each iteration $k = 0, 1, 2, \dots$, define the lower and upper bounds, respectively, over which the initial surrogate is optimized, as

$$S_k^{LB} = \begin{bmatrix} (1 - \alpha^{TR})x_k^c \\ \lfloor (1 - \alpha^{TR})x_k^d \rfloor \end{bmatrix} \quad (3.10)$$

$$S_k^{UB} = \begin{bmatrix} (1 + \alpha^{TR})x_k^c \\ \lceil (1 + \alpha^{TR})x_k^d \rceil \end{bmatrix}, \quad (3.11)$$

where $\alpha^{TR} \in [0, 1)$ is the user-specified percentage to tighten the original bounds.

- If a reduced number of evaluated points is specified for recalibration, the points selected are those whose parameter values are within the current surrogate upper and lower bounds defined by the trust region; *i.e.*,

$$\hat{X}_k = \left\{ y \in X_k : S_k^{LB} \leq \begin{bmatrix} y^c \\ y^d \end{bmatrix} \leq S_k^{UB} \right\}. \quad (3.12)$$

The overall goal of this trust region approach is the possible improvement of surrogate performance by reducing the size of the feasible region over which the surrogate problem is optimized. An illustration of this approach is given in Figure 3.3. The parameter values of the red point do not lie within the trust region; therefore, this point is not evaluated during the surrogate optimization. If a reduced number of points is specified for recalibration, it would also not be used in the construction of the surrogate.

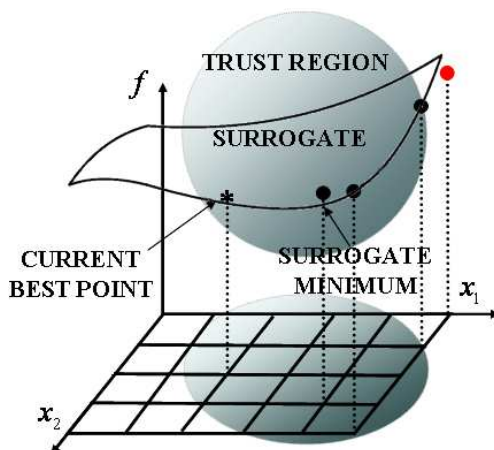


Figure 3.3 Trust Region Approach Applied to Surrogates

3.5 Implementation

The ideas presented in this chapter for solving (1.1) were incorporated in the Time-Adaptive MVMADS (TA-MVMADS) algorithm presented in Figures 3.4–3.5. The algorithm executes MVPS or MVMADS with surrogates based on function values, but with function values ordered based on a computational time based surrogate. The condition number, $\kappa(R)$ or $\kappa(\hat{\phi})$, of the function value based surrogate is monitored each time the surrogate is constructed. An updating scheme for the time cut parameter is applied using the computational times of the objective function internal processes and the associated current incumbent solution z_k . A scheme is also applied for implementing the trust region approach as specified by the user. The algorithm will be applied to three test problems in the next chapter.

TA–MVMADS ALGORITHM
INITIALIZATION AND SEARCH STEP

• **INITIALIZATION:**

Given an initial set of points X_0 , set $t_0^{cut} = \infty$, and let $x_0 \in X_0$ be the current best solution with associated function value z_0 and CPU time t_0 . Set the initial mesh M_0 satisfying (2.55) for given parameters $\Delta_0^m \leq \Delta_0^p$, D , τ and w . Set $\xi > 0$, surrogate condition number threshold κ , trust region surrogate bound update percentage $\alpha^{TR} \in [0, 1)$, and time cut constant multiplier $\alpha^{cut} > 1$. Set $k = 0$.

• **SEARCH step:**

If a trust region is not used, set $\alpha^{TR} = 0$. Set the surrogate bounds, S_k^{LB} and S_k^{UB} , based on (3.10)–(3.11), and choose surrogate recalibration points $\hat{X}_k \subseteq X_k$ based on (3.12).

Construct $F(x)$ and $T(x)$ from \hat{X}_k using the highest order polynomial less than or equal the specified order for the regression model.

If $\kappa(R) > \kappa$ or $\kappa(\hat{\phi}) > \kappa$, proceed to POLL.

Solve $\min F(x)$, yielding a finite set of trial points S_k .

For each $y \in S_k$ with $y^d \notin \Omega^d$, replace $y \in S_k$ with $\{(y^c, \lfloor y^d \rfloor), (y^c, \lceil y^d \rceil)\}$.

Evaluate $T(y)$ at each $y \in S_k$ and let \hat{S}_k be the set S_k re-ordered in ascending order of $T(y)$.

For each $y \in \hat{S}_k$ (in order), evaluate $[z, t] = f(y, t_k^{cut})$

If $z < z_k$, an improvement has been found.

Set $k = k + 1$, $x_k = y$, $z_k = z$, $t_k = t$, $t_k^{cut} = \alpha^{cut} t_k$.

Update Δ_k^m (2.50) and Δ_k^p (2.59).

Return to SEARCH.

Figure 3.4 TA–MVMADS Algorithm

TA–MVMADS ALGORITHM

POLL, EXTENDED POLL, AND CONVERGENCE

- POLL step:

Evaluate f_Ω at points in $P_k(x_k) \cup \mathcal{N}(x_k)$ until an improvement has been found or all points have been evaluated. Specifically, for each $y \in P_k(x_k) \cup \mathcal{N}(x_k)$, evaluate $[z, t] = f(y, t_k^{cut})$,

If $z < z_k$, an improvement has been found.

Set $k = k + 1$, $x_k = y$, $z_k = z$, $t_k = t$, $t_k^{cut} = \alpha^{cut} t_k$.

Update Δ_k^m (2.50) and Δ_k^p (2.59).

Return to SEARCH.

If all POLL points have been evaluated unsuccessfully and the EXTENDED POLL trigger condition (2.57) is met, proceed to EXTENDED POLL step.

Else, set $k = k + 1$, update Δ_k^m (2.51) and Δ_k^p (2.59), and proceed to SEARCH.

- EXTENDED POLL step:

Choose $\xi_k \geq \xi$ and evaluate f_Ω at points in $\mathcal{X}_k(\xi_k)$ (2.58) until an improvement has been found or all points in $\mathcal{X}_k(\xi_k)$ have been evaluated. Specifically, for each $y \in \mathcal{X}_k(\xi_k)$, evaluate $[z, t] = f(y, t_k^{cut})$,

If $z < z_k$ an improvement has been found.

Set $k = k + 1$, $x_k = y$, $z_k = z$, $t_k = t$, $t_k^{cut} = \alpha^{cut} t_k$.

Update Δ_k^m (2.50) and Δ_k^p (2.59).

Return to SEARCH.

If all EXTENDED POLL points have been evaluated unsuccessfully, set $k = k + 1$, update Δ_k^m (2.51) and Δ_k^p (2.59), update X_k with all evaluated trial points, and proceed to SEARCH.

Figure 3.5 TA–MVMADS Algorithm Continued

4. Implementation and Results

This chapter focuses on the implementation of the TA–MVMADS algorithm presented in Figures 3.4–3.5 of Section 3.5 and its numerical performance on three test problems. For each problem, different variants of the algorithm are compared to two base cases. The different variations of TA–MVMADS apply either the MVPS (see Figure 2.7) or MVMADS (see Figure 2.12) algorithm as the optimization methods, and either DACE or RBF surrogates (with or without a trust region) in the SEARCH step. The base cases apply the same optimization algorithm, but with no surrogate functions and either a single initial point or a set of CCD points (see Figure 3.2).

4.1 Coding and Processing

The TA–MVMADS algorithm was executed on a Linux operating system using three MATLAB[®] software packages: NOMADm [10] implementation of MVPS and MVMADS, DACE [58] for constructing kriging surrogates, and RBF [10] for radial basis function surrogates. The NOMADm software requires several input files for defining the following: objective and constraint functions, variable bounds, initial points, problem-specific parameters, and discrete neighbor function.

NOMADm also allows a file for defining a customized SEARCH step, which is how the surrogates were implemented in this work. Specifically, this file constructs, recalibrates, and optimizes the surrogate functions used within the overall optimization process. The optimization of the surrogate problem is performed at each SEARCH step using a recursive call to the NOMADm optimizer. During the optimization of the surrogate, the discrete variables x^d are treated as continuous variables because the DACE and RBF packages cannot handle discrete variables. The finite number of trial points S_k returned by the SEARCH step are monitored, and

if any $y^d \notin \Omega$, the values are transformed according to the rules described in Section 3.2. The time-based surrogate is then used to order the evaluation of resulting points by the true objective function. The SEARCH step surrogate optimization process also requires input files, similar to those required for the original problem. All of these are provided in Appendix B.

4.2 Test Problem 1: Lid-Driven Cavity

The first test problem is known as the *Lid-Driven Cavity* problem. This problem has long been used to test or validate new codes or new solution methods. The standard test problem is represented by a two-dimensional cross section of fluid in a box with Dirichlet boundary conditions on all sides, three of which are stationary, and the remaining one is moves with velocity tangent to the side [44]. Given a two-dimensional square domain, the Navier-Stokes equations (1.2)–(1.4) describe the fluid flow properties. Initially, the fluid is at rest, and at $t = 0$, a constant velocity is applied tangentially along one of the edges. This causes the formation of a large circular pattern of flow, known as an *eddy* [46]. At different *Reynolds* number values and simulation length times, velocity and viscosity of the fluid cause eddies to form in different patterns throughout the flow region. The different patterns for the flow properties of this problem are shown in Figure A.1 of Appendix A. For a specific *Reynolds* number, simulation length time, and simulation grid fidelity, the simulation output data can be captured and converted to image data. The image data then has noise deliberately added in order to create a reference image that represents actual experimental data. The goal of the optimization process is to systematically perform a sequence of simulations and image registrations at different *Reynolds* numbers, simulation length times, and simulation grid fidelities, and compare the resulting template and reference images of a certain flow property, in an attempt to recover the model parameters (*Reynolds* number and simulation length) at the minimum simulation fidelity. For this problem the *Heat Transfer* flow prop-

erty, as illustrated in Figures 1.4 and 1.5, respectively, is used to compare reference and template images.

4.2.1 Lid-Driven Cavity Results–MVPS

The results of each test case are shown in Table 4.1. The first two are the base cases with no SEARCH step performed. The first case uses an initial point based on the geometric center of the continuous domain and a user-specified value of $p = 40$ for the $m \times m$ “grid size” (*i.e.*, discrete (computational) parameter). The second case uses an inscribed CCD to generate the set of initial continuous parameters, while using the same user-specified value of the computational parameter. The next six cases are the three different variants of the TA–MVMADS algorithm, with and without applying a trust region. The search type labeled DACE/RBF is a SEARCH step that randomly selects which type of surrogate to employ, DACE or RBF, at each iteration. The time cutoff parameter used for all test problems is updated at each iteration according to (3.2), with a value of $\alpha^{cut} = 2$ specified as the multiplier (which is the same value used by Magallanez [60]). Any value of $\alpha^{cut} > 1$ can be specified to allow for fluctuations in the computational time of the CFD and the image registration process and give flexibility to the time cutoff parameter. For all variations of the algorithm, certain measures of algorithm performance were collected to include: objective function value, optimal values of the model and computational parameters, number of iterations, number of function evaluations, and the overall CPU time to achieve the solution.

Table 4.1 MVPS: Lid-Driven Cavity Results

Time Cut With No Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
None	Center	0.0411	500.875	5.0938	50	45	154	27.004
None	CCD	0.0411	500.875	5.0938	50	45	162	22.973
DACE	CCD	0.0411	500.878	5.0942	50	84	157	11.032
RBF	CCD	0.0411	500.877	5.0936	50	29	149	10.573
DACE/RBF	CCD	0.0411	500.877	5.0938	50	37	165	12.137
Time Cut With Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
DACE	CCD	0.0411	500.878	5.0942	50	29	143	7.999
RBF	CCD	0.0411	500.877	5.0936	49	36	164	8.629
DACE/RBF	CCD	0.0411	500.877	5.0938	50	34	196	12.973
Solution		0.00	500.136	4.9968	50	-	-	-

In each test case, roughly the same solution was found, which was sufficiently close to the true solution. The DACE surrogate with a trust region showed the fastest convergence to the solution, and the RBF surrogate was only about 30 seconds slower. Without a trust region, RBF was 30 seconds faster than DACE. Most importantly, the worst test variant (DACE/RBF) found the solution in approximately 50% less time than required by the best base case, whereas the DACE surrogate with a trust region found the solution in approximately 66% less time. These results were also generated in 93% less time than the previous results of Magallanez [60]. Some improvement was expected, since Magallanez [60] held the values for computational parameters constant instead of including them as variables to be optimized. The inclusion of simulation fidelity as an optimization variable results in a reduction in overall computation time.

Figure 4.1 is a visual representation of the solution, where the top left image is the reference image, the top middle image is the template image captured at the parameter values found by the DACE surrogate with a trust region, the top right image is the warped image after the image registration process, the bottom left is the pixel-by-pixel difference between the warped and reference images, the bottom middle shows the forces required in the x and y directions to warp the template

image, and the bottom right image is the objective function value computed using the L_2 -norm of the image differences described by (1.20)–(1.21).

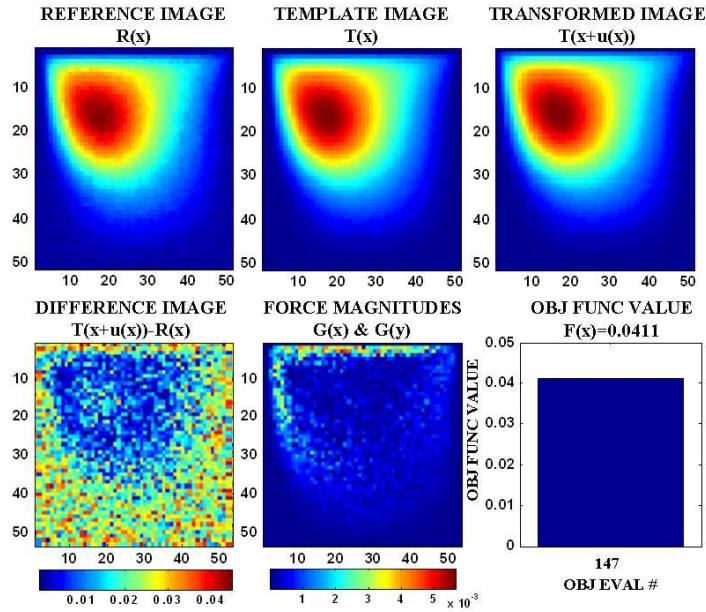


Figure 4.1 MVPS Lid-Driven Cavity Image Results

Tables 4.2 and 4.3 show more detailed performance statistics. Table 4.2 shows for each type of surrogate strategy (with and without applying a trust region), the number of SEARCH step surrogates implemented with the number and percentage of successful steps, while Table 4.3 shows the number and percentage of successful iterations performed by the SEARCH, POLL, discrete neighbor, and EXTENDED POLL steps. For clarification purposes, the percentages found in Table 4.3 and similar tables for all test problems are calculated using the number found by the current step divided by the number of incumbents found less the number found by all previous steps; *i.e.*,

$$\% \text{ Poll} = \frac{\# \text{ Poll}}{(\# \text{ Incumbents} - \# \text{ Search})} .$$

Table 4.2 MVPS: Lid-Driven Cavity Surrogate Performance

Time Cut With No Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
None	Center	-	-
None	CCD	-	-
DACE	CCD	35	8(22.8%)
RBF	CCD	29	5(17.2%)
DACE/RBF	CCD	37	10(27.1%)
Overall		101	23(22.8%)
Time Cut With Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
DACE	CCD	29	12(41.4%)
RBF	CCD	36	6(16.7%)
DACE/RBF	CCD	34	7(20.6%)
Overall		99	25(25.3%)

Table 4.3 MVPS: Lid-Driven Cavity Iteration Performance

Time Cut With No Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
None	Center	32	-	15(46.8%)	16(94.1%)	0(0.0%)
None	CCD	32	-	15(46.8%)	16(94.1%)	0(0.0%)
DACE	CCD	17	8(47.1%)	4(44.4%)	4(80.0%)	0(0.0%)
RBF	CCD	16	5(31.3%)	6(54.5%)	3(60.0%)	0(0.0%)
DACE/RBF	CCD	24	10(41.7%)	11(78.6%)	2(66.7%)	0(0.0%)
Overall		121	23(19.0%)	52(53.1%)	41(89.1%)	0(0.0%)
Overall (Surrogates)		57	23(40.4%)	22(64.7%)	9(75.0%)	0(0.0%)
Time Cut With Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
DACE	CCD	16	12(75.0%)	2(50.0%)	1(50.0%)	0(0.0%)
RBF	CCD	23	6(26.1%)	4(23.5%)	11(84.6%)	1(50.0%)
DACE/RBF	CCD	21	7(33.3%)	5(35.7%)	7(77.8%)	1(50.0%)
Overall		60	25(41.7%)	11(31.4%)	19(79.2%)	2(40.0%)

Tables 4.1–4.3 illustrate the importance of employing surrogates. As seen in Table 4.3, the SEARCH surrogates were successful in finding the incumbent solution approximately 40%–41% of the time, and when invoked, the discrete neighbor poll was successful approximately 75%–79% of the time. These percentages, along with the computational time results of Table 4.1, show that surrogate employment and the use of mixed variables are crucial in reducing overall computational time. Figure 4.2 shows the relationship between objective function values and CPU times. The first figure shows the change in objective function value and CPU time throughout the iterations; the second compares objective function value to computational time.

In the second figure, the objective function value initially decreases as the computational time decreases, followed by fluctuations in CPU time. This is caused by increases in the fidelity of the computational parameters and changes in the adaptive time cut parameter of the algorithm as the optimal solution is approached. This gives an idea of how the search path illustrated by the curved dotted path in Figure 1.2 is achieved. Other figures showing the surrogate construction and problem performance history associated with the best solution are shown in Figures A.2–A.3 of Appendix A.

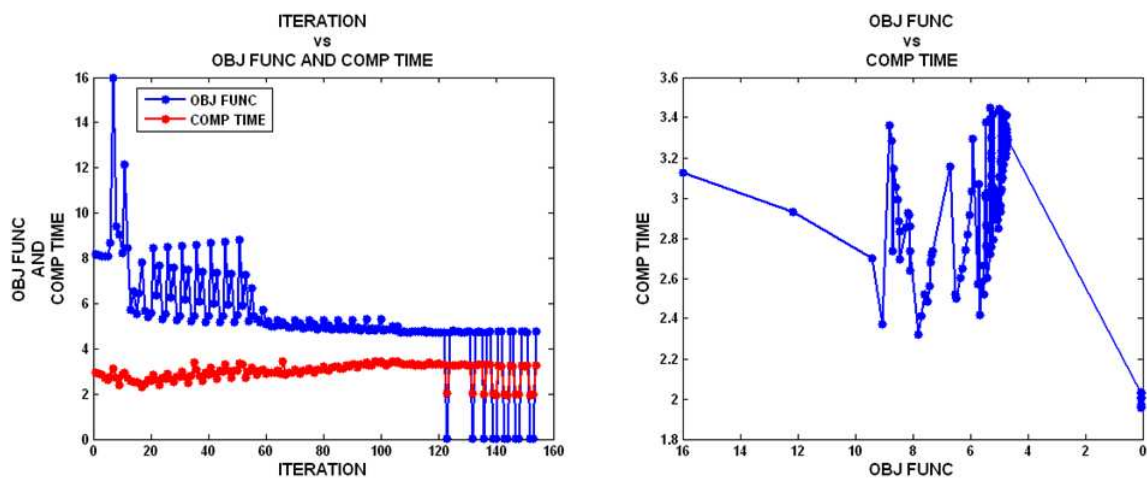


Figure 4.2 MVPS Lid-Driven Cavity Time vs Obj. Function

4.2.2 Lid-Driven Cavity Results–MVMADS

The results of each test case, when MVMADS is implemented, are shown in Table 4.4. These results are similar to those of MVPS, but the computational times increased modestly. Because of the random nature of the POLL step in selecting points for evaluation, multiple runs (10) were performed, each run yielding essentially the same results. The best run of each of the test cases is presented.

Table 4.4 MVMADS: Lid-Driven Cavity Results

Time Cut With No Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
None	Center	0.041187	500.8750	5.0195	50	58	199	32.190
None	CCD	0.041187	500.8750	5.0195	50	71	211	41.155
DACE	CCD	0.041182	500.8601	5.0066	50	88	334	16.109
RBF	CCD	0.041080	500.8460	5.0185	50	42	267	12.181
DACE/RBF	CCD	0.041183	500.8602	5.0064	50	71	424	19.182
Time Cut With Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
DACE	CCD	0.041182	500.8601	5.0066	50	57	255	15.883
RBF	CCD	0.041080	500.8460	5.0185	50	46	270	11.804
DACE/RBF	CCD	0.041183	500.8602	5.0064	50	47	314	16.736
Solution		0.00	500.1360	4.9968	50	-	-	-

Contrary to the results of MVPS, the RBF surrogate, with or without a trust region, was approximately 4 minutes faster than DACE in finding the optimum point. Again, the key performance measure is the difference in the computational times between the base cases and the variants using surrogates. The best run of the worst test variant (DACE/RBF) found the optimal solution in approximately 40% less time than required by the best base case. The RBF surrogate with a trust region found the optimal solution in approximately 63% less time. Furthermore, these results show an 84% reduction in time as compared to Magallanez [60], who did not include any computational parameters in the optimization. A visual representation is omitted because it looks identical to Figure 4.1, with the exception of the final iteration (at which the optimal solution was found).

Tables 4.5 and 4.6 show the surrogate performance and overall iteration performance parameters of the test cases using the MVMADS as the optimization method.

Table 4.5 MVMADS: Lid-Driven Cavity Surrogate Performance

Time Cut With No Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
None	Center	-	-
None	CCD	-	-
DACE	CCD	88	32(36.4%)
RBF	CCD	42	10(23.8%)
DACE/RBF	CCD	71	14(19.7%)
Overall		201	56(27.9%)
Time Cut With Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
DACE	CCD	57	17(29.8%)
RBF	CCD	46	12(26.1%)
DACE/RBF	CCD	47	5(10.6%)
Overall		150	34(22.7%)

Table 4.6 MVMADS: Lid-Driven Cavity Iteration Performance

Time Cut With No Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
None	Center	43	-	10(23.3%)	32(96.9%)	0(0.0%)
None	CCD	31	-	14(45.2%)	16(94.1%)	0(0.0%)
DACE	CCD	53	32(60.4%)	4(19.0%)	16(94.1%)	0(0.0%)
RBF	CCD	20	10(50.0%)	6(60.0%)	4(100.0%)	0(0.0%)
DACE/RBF	CCD	36	14(38.9%)	13(59.1%)	8(88.9%)	0(0.0%)
Overall		183	56(30.6%)	47(37.0%)	76(95.0%)	0(0.0%)
Overall (Surrogates)		109	56(51.4%)	23(43.4%)	28(93.3%)	0(0.0%)
Time Cut With Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
DACE	CCD	34	17(50.0%)	4(23.5%)	12(92.3%)	0(0.0%)
RBF	CCD	24	12(50.0%)	8(66.7%)	4(100.0%)	0(0.0%)
DACE/RBF	CCD	21	5(23.8%)	6(37.5%)	9(90.0%)	0(0.0%)
Overall		79	34(43.0%)	18(40.0%)	25(92.6%)	0(0.0%)

As seen in Table 4.6 the SEARCH surrogates were successful in finding the incumbent solution approximately 43%–52% of the time, and when invoked, the discrete neighbor poll was successful approximately 93% of the time. This, once more, demonstrates the significance of surrogate employment and mixed variables in the reduction of overall computational time. A figure similar to Figure 4.2 of Section 4.2.1 comparing objective function value and computational time is illustrated in Appendix A, Figure A.4. Similar results can be deduced concerning objective function value, computational time and the fluctuations as a relationship to the search path

of Figure 1.2. The surrogate response surfaces and problem performance history associated with the best solution are given in Figures A.5–A.6 of Appendix A.

4.3 Test Problem 2: Barrier Flow

The second test problem has also been extensively investigated, both experimentally and numerically [46]. It applies the Navier-Stokes equations to fluid flow over an immersed obstacle that acts as a barrier near the top of the two-dimensional flow region (see Figure 4.3). The fluid is initially at rest. At $t = 0$, an initial force is applied to the top of the fluid flow region creating an inflow velocity in the vertical downward direction. As the force moves the fluid past the immersed obstacle, the structure of the fluid flow undergoes fundamental changes. For different values of model parameters (*Reynolds* number and simulation length), the vertical and horizontal motion of the fluid can achieve different states, and the behavior can be compared. For example, at lower *Reynolds* numbers, the fluid separates prior to passing the obstacle, whereas, for higher *Reynolds* numbers, the friction along the surface of the obstacle is not strong enough to reunite the flow segments immediately; rather, the segments re-emerge further downward in the flow field.

Like the previous problem, *Reynolds* number and simulation length are the model parameters, and grid fidelity is the computational parameter to be optimized. For this problem, *Horizontal Velocity* is the flow property used for the reference and template images. This and other flow properties are shown in Figure 4.3.

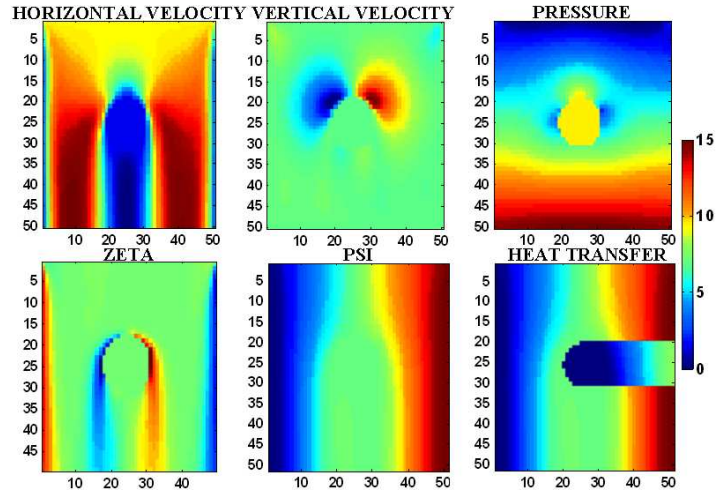


Figure 4.3 CFD Simulation Flow Properties of Barrier Flow

4.3.1 Barrier Flow Results–MVPS

The results of each test case are shown in Table 4.7. Each run was conducted in the same manner as the previous test problem, and similar results were attained with respect to the relationship between the objective function values, optimal parameter values, and computational time of the different variants. An important result is that, in all cases, the simulation grid size fidelity at the optimal solution was lower (36–41) than what was used to create the reference image (50). This gives evidence of the ability to recover the optimal model parameter values at the minimal simulation fidelity.

Table 4.7 MVPS: Barrier Flow Results

Time Cut With No Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
None	Center	0.1275	201.734	31.9844	38	40	97	45.387
None	CCD	0.1245	201.687	29.7230	41	30	94	40.103
DACE	CCD	0.1152	201.635	30.9840	36	28	138	22.032
RBF	CCD	0.1219	201.647	29.5926	36	31	172	28.338
DACE/RBF	CCD	0.1221	201.655	29.1637	36	26	146	29.564
Time Cut With Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
DACE	CCD	0.1152	201.635	30.9840	36	34	149	17.327
RBF	CCD	0.1219	201.647	29.5926	36	30	148	28.669
DACE/RBF	CCD	0.1221	201.655	29.1637	38	56	221	28.969
Solution		0.00	200.451	30.4260	50	-	-	-

The DACE surrogates with trust region approach showed the fastest convergence, finding the solution in approximately 17 minutes, a 57% improvement over the best base case. Among the surrogates without a trust region, DACE converged with the lowest computational time of 22 minutes. Figure 4.4 is a visual representation of the solution using *horizontal velocity* as the flow property of interest.

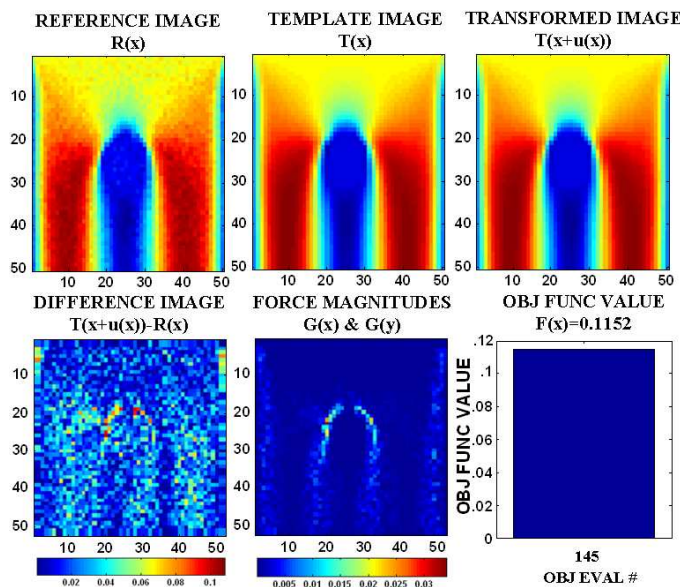


Figure 4.4 MVPS Barrier Flow Image Results

Tables 4.8 and 4.9 show the surrogate performance and overall iteration performance parameters of the test cases. The SEARCH surrogates found the incumbent solution approximately 39%–40% of the time. The discrete neighbor poll was successful approximately 50%–55% of the time when it was invoked. This percentage is lower when compared to the Lid-Driven Cavity problem, but the reduced computational times of Table 4.7 and the successes of the surrogate employments again demonstrate the value of surrogates and mixed variables for this problem.

Table 4.8 MVPS: Barrier Flow Surrogate Performance

Time Cut With No Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
None	Center	-	-
None	CCD	-	-
DACE	CCD	28	7(25.0%)
RBF	CCD	31	6(19.4%)
DACE/RBF	CCD	26	5(19.2%)
Overall		85	18(21.2%)
Time Cut With Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
DACE	CCD	34	8(23.5%)
RBF	CCD	30	6(20.0%)
DACE/RBF	CCD	56	18(32.1%)
Overall		120	32(26.7%)

Table 4.9 MVPS: Barrier Flow Iteration Performance

Time Cut With No Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
None	Center	27	-	24(88.9%)	2(66.7%)	0(0.0%)
None	CCD	17	-	15(88.2%)	1(50.0%)	0(0.0%)
DACE	CCD	15	7(46.7%)	6(75.0%)	1(50.0%)	0(0.0%)
RBF	CCD	18	6(33.3%)	10(83.3%)	1(50.0%)	0(0.0%)
DACE/RBF	CCD	13	5(38.5%)	6(75.0%)	1(75.0%)	0(0.0%)
Overall		90	18(20.0%)	61(84.7%)	6(54.5%)	0(0.0%)
Overall (Surrogates)		46	18(39.1%)	22(78.6%)	3(50.0%)	0(0.0%)
Time Cut With Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
DACE	CCD	21	8(38.1%)	11(84.6%)	1(50.0%)	0(0.0%)
RBF	CCD	17	6(35.3%)	9(81.8%)	1(50.0%)	0(0.0%)
DACE/RBF	CCD	43	18(41.9%)	23(92.0%)	1(50.0%)	0(0.0%)
Overall		81	32(39.5%)	43(87.8%)	3(50.0%)	0(0.0%)

The computational time required to solve this problem took approximately twice as long as the Lid-Driven Cavity problem. Figure 4.5, a comparison of the Barrier Flow objective function value to computational time, shows more dynamic fluctuations in computational time, compared to those observed in Figure 4.2 for the previous problem. Therefore a two-dimensional plot of the objective function versus *Reynolds* number was generated for the two problems shown in Figure 4.6 (simulation length and grid fidelity were fixed at their optimal parameter values). These plots illustrate how much more dynamic and challenging this problem is, compared to the Lid-Driven Cavity problem. Illustrations of the surrogate response surfaces and the

problem performance history associated with the best solution are given in Figures A.7–A.8, of Appendix A.

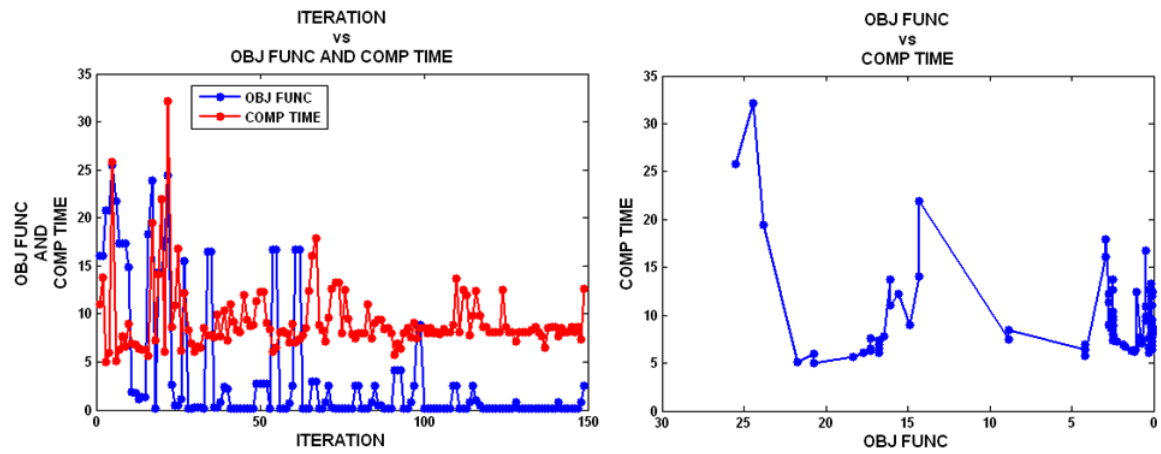


Figure 4.5 MVPS Barrier Flow Time vs Obj. Function

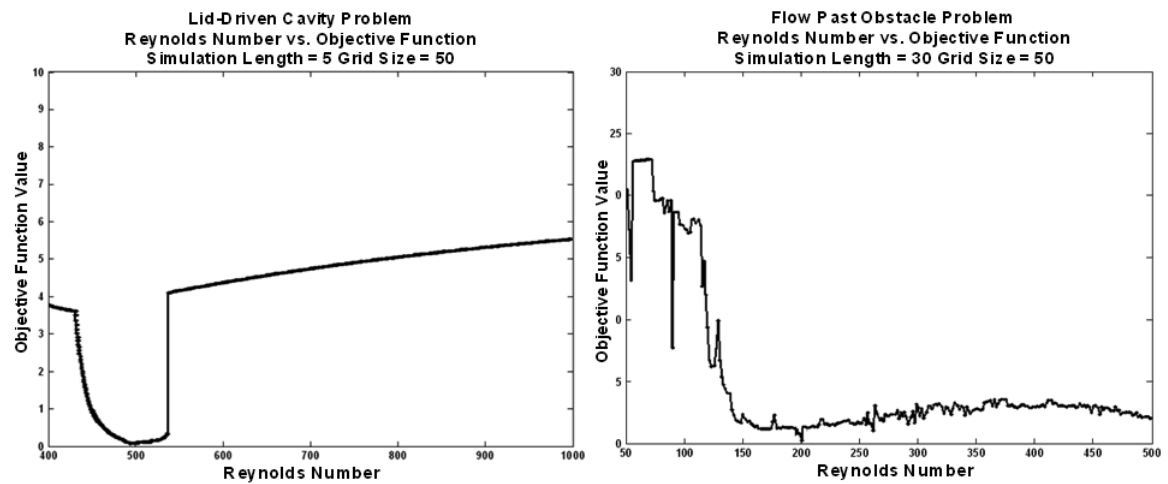


Figure 4.6 Lid-Driven Cavity and Barrier Flow Problem Mappings

4.3.2 Barrier Flow Results–MVMADS

Tables 4.10–4.12 give results and surrogate performance data for the TA–MVMADS algorithm. The results are similar to those of MVPS with respect to the optimal objective function, model, and computational parameter values. The overall

computational times required for the different variants either increased slightly or remained approximately the same. Once again, because of the random nature of the MVMADS POLL step, 10 runs were performed, and each run resulted in approximately what was shown in the application of MVPS on this problem, and the best run of each test case is presented.

Table 4.10 MVMADS: Barrier Flow Results

Time Cut With No Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
None	Center	0.1253	201.720	29.4844	38	39	123	54.561
None	CCD	0.1225	201.557	31.2768	41	60	192	48.438
DACE	CCD	0.1152	201.635	30.9645	36	28	138	26.122
RBF	CCD	0.1206	201.648	30.9840	36	31	174	28.866
DACE/RBF	CCD	0.1214	201.674	29.8871	36	32	167	36.215
Time Cut With Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
DACE	CCD	0.1152	201.635	30.9645	38	39	166	20.736
RBF	CCD	0.1206	201.648	30.9840	36	30	154	22.875
DACE/RBF	CCD	0.1214	201.674	29.8871	38	49	196	35.696
Solution		0.00	200.451	30.4260	50	-	-	-

Table 4.11 MVMADS: Barrier Flow Surrogate Performance

Time Cut With No Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
None	Center	-	-
None	CCD	-	-
DACE	CCD	28	7(25.0%)
RBF	CCD	31	6(19.4%)
DACE/RBF	CCD	32	7(21.9%)
Overall		91	20(21.9%)
Time Cut With Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
DACE	CCD	39	11(28.2%)
RBF	CCD	30	6(20.0%)
DACE/RBF	CCD	49	12(24.5%)
Overall		118	29(24.6%)

Table 4.12 MVMADS: Barrier Flow Iteration Performance

Time Cut With No Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
None	Center	43	-	10(23.3%)	32(96.9%)	0(0.0%)
None	CCD	31	-	14(45.2%)	16(94.1%)	0(0.0%)
DACE	CCD	53	32(60.4%)	4(19.0%)	16(94.1%)	0(0.0%)
RBF	CCD	20	10(50.0%)	5(50.0%)	4(80.0%)	0(0.0%)
DACE/RBF	CCD	36	14(38.9%)	13(59.1%)	8(88.9%)	0(0.0%)
Overall		183	56(30.6%)	47(37.0%)	76(95.0%)	0(0.0%)
Overall (Surrogates)		109	56(51.4%)	23(43.4%)	28(93.3%)	0(0.0%)
Time Cut With Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
DACE	CCD	26	11(42.3%)	13(86.7%)	1(50.0%)	0(0.0%)
RBF	CCD	17	6(35.3%)	9(81.8%)	1(50.0%)	0(0.0%)
DACE/RBF	CCD	36	12(33.3%)	22(91.7%)	1(50.0%)	0(0.0%)
Overall		79	29(36.7%)	44(88.0%)	3(50.0%)	0(0.0%)

The SEARCH surrogates, accounted for 36%–52% of the new incumbent solutions compared to the 39% found by MVPS on this problem. Although the discrete neighbor poll showed fewer successes for the surrogates with a trust region as compared to the MVMADS variants of the Lid-Driven Cavity problem, the overall quality of producing the incumbent when compared to the application of MVPS to this problem increased to 50%–93%. The reduction in computational time from 48 minutes for the best base case, to 20 minutes for the best surrogate variant, shows the success and significance of using mixed variables and surrogates for this problem.

A figure similar to Figure 4.5 showing the relationship between objective function values and computational times is illustrated in Appendix A, Figure A.9. Identical to the MVGPS results, the computational time required for MVMADS on this problem was approximately twice that of the Lid-Driven Cavity problem. This can be attributed mainly to the dynamic nature of the Barrier Flow problem, as discussed in Section 4.3.1. Illustrations of the surrogate response surfaces and problem performance history associated with the best solution can be seen in Figures A.10–A.11 of Appendix A.

4.4 Test Problem 3: Liquid Drop

The final problem is another classical problem first studied by Harlow and Shannon [48]. This problem applies the Navier-Stokes equations to a droplet of fluid falling into a fluid filled basin (see Figure 4.7). Given a two-dimensional cross section of the domain with dimensions $[0, a] \times [0, b]$, the fluid-filled basin occupies the lower half of this cross section. The droplet has a radius of $b/10$, is centered at $(a/2, 2b/3)$, and has an initial vertical velocity of $v_0 = -2$. As time starts, the droplet impacts the surface of the fluid, a trough is formed, and the structure of both the droplet and the fluid in the basin begin dynamically changing. The droplet forces the fluid in the basin to be displaced both vertically and horizontally. The vertical and horizontal forces displacing the fluid reach the boundaries and each reflects in the opposite direction. The droplet sloshes back up out of the basin and falls back in, causing asymmetry and instabilities. This continues until the forces in the fluid region reach a steady state [46]. As with the two previous problems, *Reynolds* number and simulation length are the model parameters and grid fidelity is the computational parameter to be optimized. The *Horizontal Velocity* is the flow property used for the reference and template images. This and other flow properties are shown in Figure 4.7.

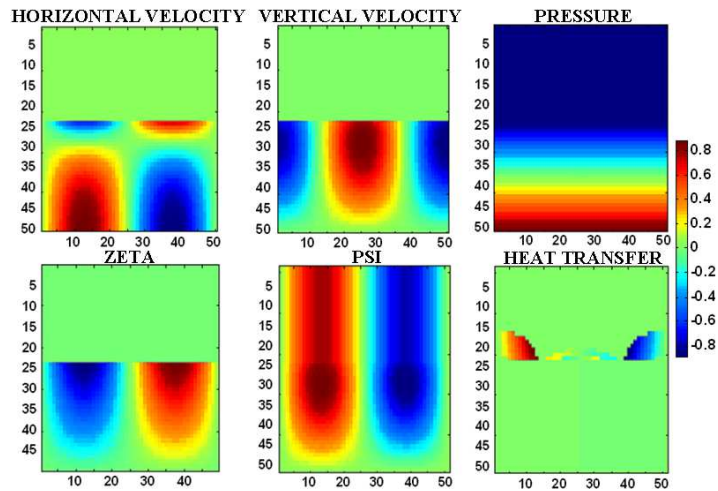


Figure 4.7 CFD Simulation Flow Properties of Liquid Drop

4.4.1 Liquid Drop Results–MVPS

Each run was conducted in the same manner as the previous test problems; however, even though the algorithm converged to parameter values close to the optimal values, it did not obtain optimal solutions, as shown in Table 4.13. The computational times of each variation increased compared to the results seen in the previous test problems. Also, the objective function and model/computational parameter values determined by the optimization process differed among the variations of the algorithm applications. Because of this and the ineffectiveness of using the default extended poll trigger value ($\xi_k = \max\{\xi, 0.01|f(x_k)|\}$) in the other problems, an additional run was performed using an increased value ($\xi_k = \max\{\xi, 0.5|f(x_k)|\}$). The results were identical for each test case except for a 75% increase in the CPU time, which is attributed to the increase in number of function evaluations incurred by the EXTENDED POLL step. These results are not shown because the results were essentially the same.

Table 4.13 MVPS: Liquid Drop Results

Time Cut With No Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
None	Center	9.2032	47.000	9.6152	41	20	72	114.217
None	CCD	9.2032	47.000	9.6152	41	20	80	111.951
DACE	CCD	3.8762	51.781	11.4976	48	36	147	44.195
RBF	CCD	4.3497	48.396	10.4637	41	38	153	57.811
DACE/RBF	CCD	4.9326	47.255	11.0162	44	37	144	73.531
Time Cut With Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
DACE	CCD	3.8197	52.875	11.6563	48	24	129	39.672
RBF	CCD	4.0172	47.367	10.4551	41	19	113	51.381
DACE/RBF	CCD	4.8689	46.117	11.2656	44	23	124	71.052
Solution		0.00	50.011	10.0160	50	-	-	-

The DACE surrogates with a trust region showed the fastest convergence, finding the best solution of all test cases in approximately 40 minutes. This was 32 and 23 minutes longer, respectively, than for the Lid-Driven Cavity and Barrier Flow problems. Without a trust region, the DACE surrogate converged to a similar

solution but required an additional 5 minutes. Figure 4.8 is a visual representation of the solution with *Horizontal Velocity* used as the flow property of interest.

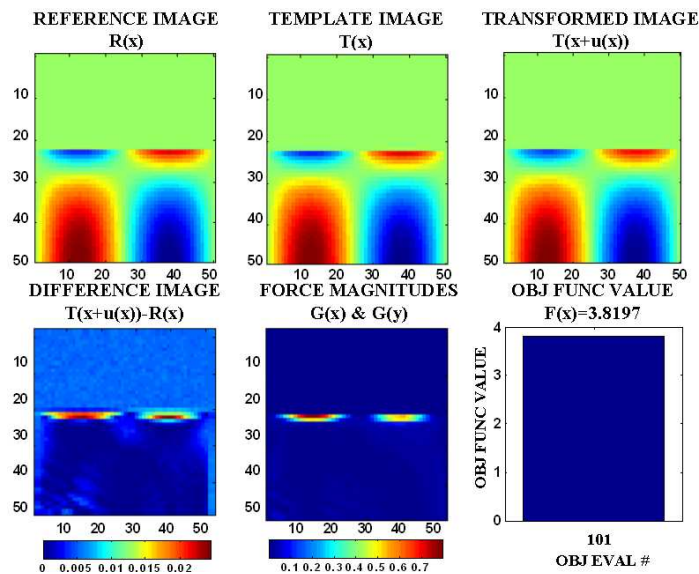


Figure 4.8 MVGPS Liquid Drop Image Results

Tables 4.14 and 4.15 show the surrogate performance and overall iteration performance parameters, respectively, of the test cases. The SEARCH surrogates found a new incumbent solution approximately 37%–42% of the iterations, but fewer overall incumbent solutions were found than for the other test problems. The discrete neighbor poll seemed to be almost ineffective, producing a total of only 3 incumbents. Even though performance measures were worse when compared to the optimal results of the previous problems, the use of surrogates resulted in roughly a 65% reduction in computational time.

Table 4.14 MVPS: Liquid Drop Surrogate Performance

Time Cut With No Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
None	Center	-	-
None	CCD	-	-
DACE	CCD	36	6(16.7%)
RBF	CCD	38	4(10.5%)
DACE/RBF	CCD	27	4(10.8%)
Overall		111	14(12.6%)
Time Cut With Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
DACE	CCD	24	7(29.2%)
RBF	CCD	19	4(21.1%)
DACE/RBF	CCD	23	5(21.7%)
Overall		66	16(24.2%)

Table 4.15 MVPS: Liquid Drop Iteration Performance

Time Cut With No Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
None	Center	7	-	5(71.4%)	1(50.0%)	0(0.0%)
None	CCD	7	-	5(71.4%)	1(50.0%)	0(0.0%)
DACE	CCD	15	6(40.0%)	7(77.8%)	1(50.0%)	0(0.0%)
RBF	CCD	11	4(36.4%)	5(71.4%)	1(50.0%)	0(0.0%)
DACE/RBF	CCD	11	4(36.4%)	5(71.4%)	1(50.0%)	0(0.0%)
Overall		51	14(27.5%)	27(72.9%)	5(50.0%)	0(0.0%)
Overall (Surrogates)		37	14(37.8%)	17(73.9%)	3(50.0%)	0(0.0%)
Time Cut With Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
DACE	CCD	16	7(43.8%)	8(88.9%)	0(0.0%)	0(0.0%)
RBF	CCD	11	4(36.4%)	6(85.7%)	0(0.0%)	0(0.0%)
DACE/RBF	CCD	11	5(45.5%)	5(83.3%)	0(0.0%)	0(0.0%)
Overall		38	16(42.1%)	19(86.4%)	0(0.0%)	0(0.0%)

Due to the increase in computational time and the less-than-optimal solutions attained when solving this problem, a two-dimensional plot of the objective function versus *Reynolds* number was generated, similar to the one described in Section 4.3.1. Specifically, Figures 4.9–4.10 show computational time and objective function value, and objective function value versus *Reynolds* number, respectively. The fluctuations in the objective function value and computational times are much more dominant in this problem than those of the previous two problems, as seen in Figures 4.2 and 4.5. This helps explain the difficulty of finding the true optimal solution and the overall increase in CPU time. Illustrations of the surrogate response surfaces and

problem performance history associated with the best solution can be seen in Figures A.12–A.13, respectively, of Appendix A.

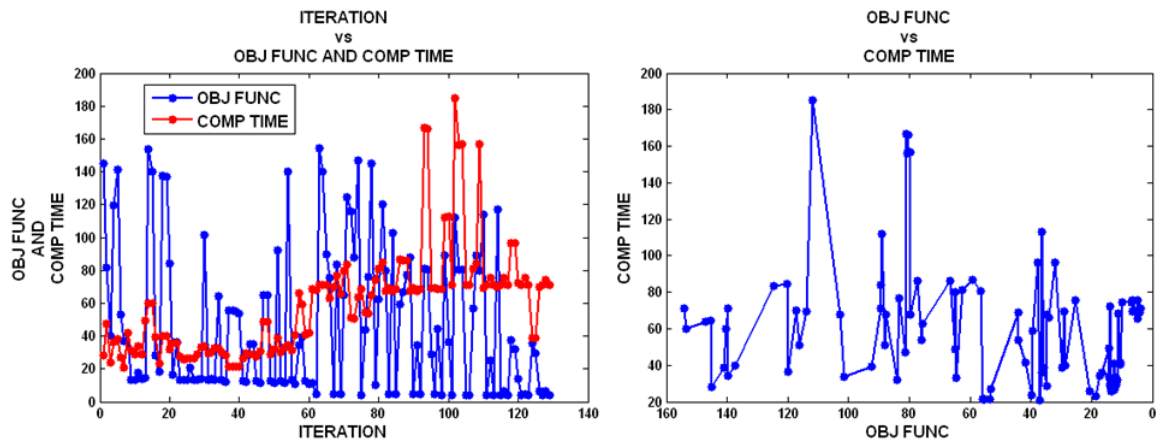


Figure 4.9 MVGPS Liquid Drop Time vs Obj. Function

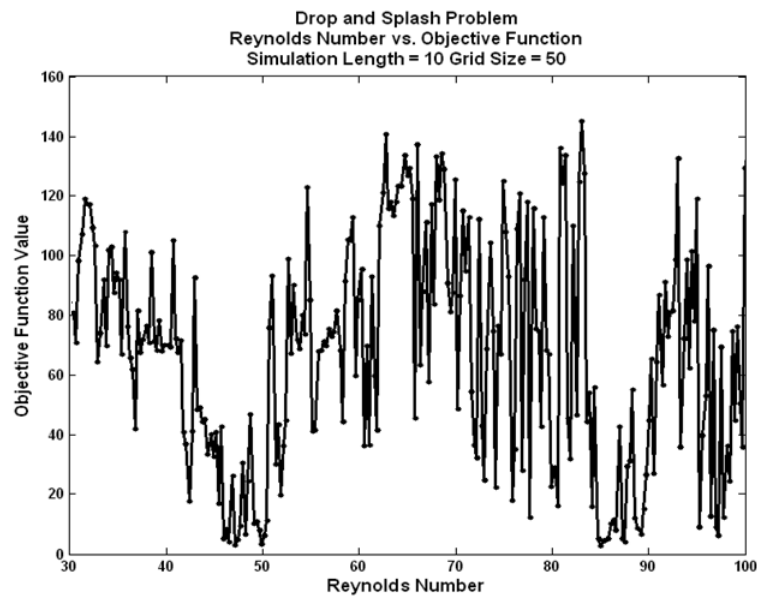


Figure 4.10 Liquid Drop Problem Mapping

4.4.2 Liquid Drop Results–MVMADS

As with MVMADS results of the other two test problems, multiple (10) runs were performed. The initial runs showed similar results, but the best of the ten

runs found a 96% reduction in optimal objective function value with a minimal grid size fidelity less than that of the reference image. This solution was found with an increase of 25 minutes in computational time over the results of MVPS. Tables 4.16–4.18 give the results and surrogate performance of the best run of the test cases. The increase in computational time is possibly attributed to the randomness of the MVMADS POLL step; however, this may be beneficial for problems that have objective functions with increased fluctuations.

Table 4.16 MVMADS: Liquid Drop Results

Time Cut With No Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
None	Center	10.4853	53.000	11.4844	41	28	98	252.784
None	CCD	9.0669	46.000	9.8828	41	36	139	215.740
DACE	CCD	4.3692	51.488	11.1660	41	35	145	166.748
RBF	CCD	2.1845	51.405	11.1055	50	38	240	116.240
DACE/RBF	CCD	2.8964	50.197	11.6291	50	35	181	118.455
Time Cut With Trust Region								
Search Type	Initial Pt(s)	Obj. Val.	Re Num.	Sim Len.	Grid Size	# of Iters.	# of Evals.	CPU Time (min)
DACE	CCD	4.1683	51.632	11.2141	41	25	140	88.254
RBF	CCD	0.1445	50.614	10.0023	44	22	142	64.228
DACE/RBF	CCD	0.2378	49.804	10.5657	41	35	228	72.796
Solution		0.00	50.011	10.0160	50	-	-	-

Table 4.17 MVMADS: Liquid Drop Surrogate Performance

Time Cut With No Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
None	Center	-	-
None	CCD	-	-
DACE	CCD	35	7(20.0%)
RBF	CCD	38	7(18.4%)
DACE/RBF	CCD	35	6(17.1%)
Overall		108	20(18.5%)
Time Cut With Trust Region			
Search Type	Initial Pt(s)	# of Search Surrogates	# (%) Success
DACE	CCD	25	6(23.9%)
RBF	CCD	22	7(31.8%)
DACE/RBF	CCD	35	7(20.0%)
Overall		82	20(24.4%)

Table 4.18 MVMADS: Liquid Drop Iteration Performance

Time Cut With No Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
None	Center	8	-	6(75.0%)	1(50.0%)	0(0.0%)
None	CCD	12	-	10(83.3%)	1(50.0%)	0(0.0%)
DACE	CCD	17	7(41.2%)	9(90.0%)	0(0.0%)	0(0.0%)
RBF	CCD	13	7(53.8%)	4(66.7%)	1(50.0%)	0(0.0%)
DACE/RBF	CCD	12	6(50.0%)	5(83.3%)	0(0.0%)	0(0.0%)
Overall		62	20(32.3%)	34(80.9%)	3(37.5%)	0(0.0%)
Overall (Surrogates)		42	20(47.6%)	18(81.8%)	1(25.0%)	0(0.0%)
Time Cut With Trust Region						
Search Type	Initial Pt(s)	# of Incumbents	# (%) (Search)	# (%) (Poll)	# (%) (Neigh.)	# (%) (Ext. Poll)
DACE	CCD	17	6(35.3%)	10(90.9%)	0(0.0%)	0(0.0%)
RBF	CCD	12	7(58.3%)	4(80.0%)	0(0.0%)	0(0.0%)
DACE(50)/RBF(50)	CCD	12	7(58.3%)	4(80.0%)	0(0.0%)	0(0.0%)
Overall		41	20(48.8%)	18(85.7%)	0(0.0%)	0(0.0%)

The SEARCH surrogates found 47%–49% of the incumbent solutions. This is an increase when compared to the performance of MVPS. The discrete neighbor poll still showed poor performance, but with respect to finding an improved solution in the least amount of computational time, the use of surrogates with a trust region shows a significant decrease in computational time, as compared to the other test cases. Figure 4.11 gives a visual representation of the solution.

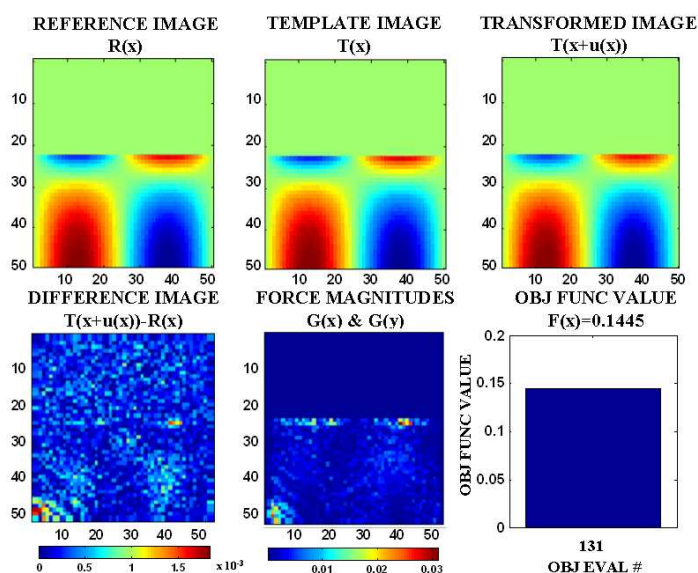


Figure 4.11 MVMADS Liquid Drop Image Results

Figure 4.12 shows similar, if not increased, fluctuations of objective function value and computational time, as compared to Figure 4.9. This demonstrates the effectiveness of the TA–MVMADS algorithm (in this case, with MVMADS as the optimization method), as applied to this class of problems, when dealing with an objective function that is dominated by fluctuations in function value and computational time. Illustrations of the surrogate response surfaces and problem performance history associated with the best solution are given in Figures A.14–A.15, respectively, of Appendix A.

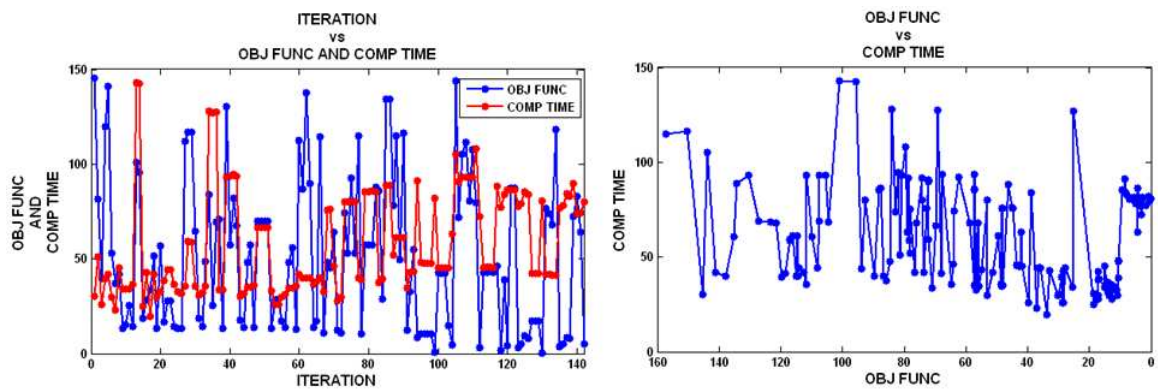


Figure 4.12 MVMADS Liquid Drop Time vs Obj. Function

4.5 Summary

The TA–MVMADS algorithm presented in Chapter 3 was applied to three test problems showing effective results. The algorithm was successful in finding the optimal solution and parameter values on the first and second problems. On the second problem, the algorithm found the optimal solution at a reduced grid size fidelity, giving evidence of the ability to recover optimal model parameters while recovering the minimal simulation fidelity. The third problem proved to be much more difficult, due to the increased fluctuations in the objective function value and computational time, but the MVMADS variant of the algorithm was able to approximately recover the optimal solution and model parameters at a reduced grid size fidelity with a modest increase in computational time. Chapter 5 offers some concluding comments and suggests some ideas for future work.

5. Conclusions and Recommendations

The techniques applied in this research provided an investigation into the improvement of computational and model parameter optimization of engineering problems. The focus was placed on the use of mixed variable optimization methods, multiple surrogate functions, and trust regions to develop an optimization framework for reducing computational expense in the recovery of optimal parameter values. Significant reductions in computational times were achieved, as compared to the only previous study [60] on this class of optimization problems. The remainder of this chapter gives concluding remarks, summarizes the contributions of this work, and suggests potential areas of improvement and future research.

5.1 *Summary and Conclusions*

The three problems studied in Chapter 4 showed different levels of complexity. This was evident in the computational times required to solve the different problems. For the first two problems of Section 4.2 and Section 4.3, the TA–MVMADS algorithm (see Figures 3.4 and 3.5), developed for use in the optimization framework illustrated in Figure 1.1, demonstrated significant improvement in the computational times required to recover the optimal parameters when compared to the previous work of Magallenez [60]. The third problem of Section 4.3 showed different results. The optimal parameters were recovered by only two variants of the MVMADS algorithm.

In order to understand the different levels of difficulty, it was important to study the possible causes of the increase in computational times from one problem to another. Therefore, a two-dimensional plot of the objective function versus the *Reynolds* number was generated for each problem (see Figures 4.6 and 4.10), while fixing the simulation length and grid fidelity at their optimal values. These plots illustrate the increased dynamic volatility and fluctuations among the problems.

The first problem's objective function was much less dynamic than that of the second problem, and both of these had significantly fewer fluctuations than that of the third problem. Surrogate strategies were implemented in an attempt to guide the optimization process to a region of the parameter space containing the optimal solution in the least amount of computational time possible. The increased volatility witnessed in the second and third problems makes it difficult to construct surrogate functions that effectively represent the behavior of the true objective function. This type of dynamic behavior in objective functions can increase the number of function evaluations required to find an optimal solution, thus causing an increase in the overall computational time, or it can lead the algorithm to a poor local solution. This is evident in the results of the third test problem. However, the MVMADS variant of the algorithm recovered the optimal solution and model parameters at a reduced grid size fidelity with only modest increases in computational time. This is possible evidence of the benefits in the randomness of an MVMADS POLL step when dealing with applications that have objective functions dominated by fluctuations.

The following list describes the contributions of this research:

- An algorithm was developed to incorporate the use of mixed variable optimization techniques for reducing the computational time required to recover optimal model parameters and minimum simulation fidelity in certain classes of engineering problems. Optimization of computational parameters implies less computational time for the simulation, faster convergence to optimal solutions, and more (but less expensive) function evaluations, meaning a more thorough yet efficient search of the parameter space.
- A new strategy was developed for making use of both objective-based and time-based surrogates. Specifically, the surrogate function $F(x)$ was optimized and a finite set of trial points were returned. The trial points were then ordered and selected for evaluation in the objective function using the time-based surrogate $T(x)$.

- Radial Basis Functions (RBF) were added to the customized SEARCH files as a class of surrogate functions, which could be compared with DACE surrogates. Another addition was the surrogate function that randomly selects DACE or RBF at each iteration. These two additions expand the availability of surrogate functions that can be applied to this class of problems. As was seen in the Chapter 4 results, multiple surrogate function types can add an effective capability to the algorithm and optimization framework.
- The methods used for construction of both DACE and RBF were improved. Specifically, a singular value decomposition (SVD) was added to the surrogate construction process to alleviate problems encountered when the correlation matrix R or the basis function matrix $\hat{\phi}$ become ill-conditioned. As the optimal objective function value is approached and points are added to the initial surrogate for recalibration, the points begin clustering, resulting in R and $\hat{\phi}$ becoming ill-conditioned [25]. Singular value decomposition makes it possible to approximate the inverse of R or $\hat{\phi}$ instead of computing it directly.
- The methods used in this research improved the effectiveness of the computational time threshold parameter t_k^{cut} . This parameter now uses the overall computational time of an objective function evaluation; *i.e.*, both the CFD simulation time and the image registration time associated with the current best solution are used for updating the value of t_k^{cut} (3.2). This is more efficient because the CFD simulation times change due to increases/decreases in grid size fidelity and dynamic behavior of the problem.
- A trust region approach was implemented in the algorithm for setting and updating the upper and lower bounds for the surrogate optimization problem. This approach reduces the feasible region over which the surrogate functions are optimized. It can also control the points used for the recalibration of the surrogate functions. This improves surrogate performance by constructing and optimizing the surrogate in a region of the parameter space showing

improvement in objective function value. This also helps alleviate incorrect trends or behaviors and reduce the ill-conditioning of the correlation matrix or basis function matrix that can occur when the surrogate is constructed and optimized over the entire feasible region and known design sites.

- Numerical results were generated for the three test problems, and DACE and RBF surrogate functions were shown to perform similarly. That is, there is no significant evidence of one outperforming the other, even in the presence or absence of a trust region. The successful performance in Chapter 4 of the TA–MVMADS algorithm demonstrates the effectiveness of the algorithm in certain classes of engineering problems.

5.2 Future Areas of Research

The use of different methods to develop a new technique for solving this class of problems has provided potential areas of improvement and future research in the use of mixed variable parameter optimization in engineering problems. These topics of interest are now discussed.

Different Methods for Generating Initial Points. A CCD is not typically used for problems of higher dimensionality because, as the number of variables to be optimized increases, it requires more function evaluations, which makes a CCD inefficient for computationally expensive problems. During the first stages of this research, the original domain associated with continuous variables was very large, and a CCD was used because it generated better surrogates. In this domain, a “space-filling” sampling would require an increased number of design sites to extract information over this domain. However, the domain was reduced, based on expert knowledge of the problems [16], and Latin hypercube sampling, orthogonal arrays, or nearly uniform designs [59] could prove to be more effective than a CCD at gaining maximum information using a minimal number of sample points.

These “space-filling” methods can be applied iteratively and possibly be used by the SEARCH step, thus giving more capability to the user. The NOMADm software and TA–MVMADS algorithm are currently capable of implementing multiple types of CCDs, Latin hypercube samplings, and nearly uniform designs as experimental design methods.

A Method for Adapting the α^{cut} Multiplier. The t_k^{cut} parameter used for all test problems was updated at each iteration according to (3.2), and a value of $\alpha^{cut} = 2$ was specified as the constant multiplier. Any value of $\alpha^{cut} > 1$ can be specified to allow for fluctuations in the computational time of the CFD and the image registration process and give flexibility to the t_k^{cut} parameter. The value of $\alpha^{cut} = 2$ was chosen because it was used by Magallanez [60]. A method for adapting the value of α^{cut} could be developed based on the cached history of values of the t_k^{cut} parameter. For example, if the value of t_k^{cut} has increased over several iterations, moderately decrease the value of α^{cut} to obtain a more restrictive t_k^{cut} parameter, and if it has decreased, possibly alter α^{cut} in an opposite manner to obtain a less restrictive t_k^{cut} parameter. A test of other values of α^{cut} , as applied to these test problems, would help determine if a method of this type could be effective.

Adapting the Order of the Regression Polynomial Used for DACE and RBF Surrogates. Both surrogate types used a second-order polynomial as the highest order for the regression model. A higher order polynomial may have improved the effectiveness of the surrogate, but the degree of the polynomial determines how many design points are needed to construct the surrogate. A higher order polynomial requires more design points, thus increasing the number of evaluations and computational time. In this work, a high order polynomial could be specified, but the algorithm would choose the order of the polynomial it could construct based on the available number of design points, not to exceed the order specified by the user. Therefore, lower-order polynomials could be used until enough design points were available to construct the desired polynomial.

A more robust method for adapting the order of the regression polynomial could be developed. A method that allows the algorithm to adapt the order of the polynomial based on performance history throughout the optimization process would add flexibility and possibly increase the effectiveness of surrogates.

Adapting DACE and RBF Correlation and Basis Function Surrogates. The choice of a *Gaussian* correlation model (see (2.33)) was based on a suggestion in [57], which describes it as a common choice used in practice for physical phenomena, and which shows similar behavior to the desired function as the number of points increases. There are other models available (see (2.33)), such as *Cubic*, *Spline*, *Exponential*, *Linear*, and *Spherical*, one of which may prove to be more effective at approximating the underlying behavior of objective functions for this class of problems. Similarly, the choice of a *Multiquadric* basis function (see (3.9)) was based on a numerical study of scattered data interpolation, which found that this basis function provided the most accurate interpolation surfaces of all the basis function forms for interpolation in two dimensions [43]. It is possible that one of the other basis function forms in (2.37)–(2.41) may produce more accurate RBFs, but no other forms were tested.

This research did not use any methods for adapting the type of correlation model or basis function during the optimization. Just as regression models can be adapted, the type of correlation model or basis function could be changed based on performance history. This could help reduce the possible dynamic behavior of the surrogate response surfaces by choosing the correlation model or basis function that more accurately represents underlying trend behavior of the true objective function.

Methods to Alleviate Ill-Conditioning. Singular value decomposition was used to reduce the problem of an ill-conditioned correlation matrix or basis function. This method is computationally expensive when the number of design sites increases. An alternative approach suggested by Booker [25] is to model the output as the sum

of two Gaussian independent processes, one with correlation parameters based on the initial set of points, and one based on subsequent points.

More Robust Trust Region Approach. The trust region approach applied here is based on the “*trust radius*” or “*move limit*” ideas presented in [13]. The trust region is applied to the surrogate based on the current best feasible solution, and the upper and lower bounds over which the surrogate is optimized are tightened by a user-specified percentage α^{TR} of the original bounds. These surrogate bounds can change at each iteration. Though fairly simple, this approach proved to be effective in increasing the surrogate performance by constructing and optimizing the surrogate in a region of the parameter space showing improvement in objective function value. The points used for the recalibration of the surrogate functions are also controlled. This helps alleviate incorrect trends or behaviors and reduce the ill-conditioning of the correlation model and basis function. There are, in fact, more robust methods for applying trust regions to manage approximation models [13, 39] that could be applied. Another approach, mentioned in [60], is a trust region based on the current frame size Δ_k^p or mesh size Δ_k^m , where the relationship between the number of points within a region and the distance between the points would determine the size of the trust region.

Customized Discrete Neighbor Set. For the purpose of this study, the discrete neighbor set was a default neighborhood (2.52), in which the continuous variables are held constant and one discrete variable is changed at a time by a single unit. Other neighbor sets are certainly allowed and may, in fact, improve performance. For example, it could be beneficial to allow the discrete variable to change by $n > 1$ units at a time – perhaps where n is chosen randomly.

Increase the Dimensionality of the Test Problems. In order to evaluate the effectiveness of the TA–MVMADS algorithm more thoroughly, classical test problems that are inherently more difficult could be used. If these types of test

problems are not readily available, the number of parameters to be optimized (both continuous and discrete) can be increased.

The results of this study suggest the need for future research in improving mixed variable optimization techniques involving computational and model parameters in engineering problems, and the list just described is by no means exhaustive.

Bibliography

1. Abramson, M. A. *Pattern Search Algorithms for Mixed Variable General Constrained Optimization Problems*. Ph.D. thesis, Department of Computational and Applied Mathematics, Rice University, August 2002.
2. Abramson, M. A., C. Audet, and J. E. Dennis, Jr. “Optimization using Surrogates for Engineering Design”. *IMA Postdoctoral Fellowship Lecture Series*, <http://www.ima.umn.edu/talks/workshops/dennis/JuniorUMN.pdf>, 2003.
3. Abramson, M. A. “Mixed Variable Optimization of a Load-Bearing Thermal Insulation System using a Filter Pattern Search Algorithm”. *Optimization and Engineering*, 5:157–177, 2004.
4. Abramson, M. A. “Second-Order Behavior of Pattern Search”. *SIAM Journal on Optimization*, 16(2):315–330, 2005.
5. Abramson, M. A., and C. Audet. “Convergence of Mesh Adaptive Direct Search to Second-Order Stationary Points”. *SIAM Journal on Optimization*, 17(2):606–619, 2006.
6. Abramson, M. A., C. Audet, and J. E. Dennis, Jr. “Nonlinear Programming by Mesh Adaptive Direct Searches”. *SIAG/Optimization Views-and-News*, 17(1):2–11, 2006.
7. Abramson, M. A., C. Audet, and J. E. Dennis, Jr. “Filter Pattern Search Algorithms for Mixed Variable Constrained Optimization Problems”. *Pacific Journal of Optimization*, 3(3):477–500, 2007.
8. Abramson, M. A., C. Audet, J. W. Chrissis, and J. G. Walston. “Mesh Adaptive Direct Search Algorithms for Mixed Variable Optimization”. *GERAD Technical Report 2007-47*. *Optimization Letters*, to appear.
9. Abramson, M. A. “Mesh Adaptive Direct Search for Derivative-Free Optimization”. *Los Alamos National Laboratory Data Sciences Seminar*, 2007.
10. Abramson, M. A. “NOMADm optimization software”. <http://www.afil.edu/en/ENC/Faculty/MAbramson/NOMADm.html>, 2007.
11. Abramson, M. A. “NOMADm version 4.5 Users’ Guide”. <http://www.afil.edu/en/ENC/Faculty/MAbramson/NOMADm.html>, 2007.
12. Abramson, M. A., T. J. Asaki, J. E. Dennis, Jr., K. R. O’Reilly, and R. L. Pingel. “Quantitative Object Reconstruction Via Abel-based X-ray Tomography and Mixed Variable Optimization”. Technical Report TR07-03, Rice University, Department of Computational and Applied Mathematics (CAAM), 2007.

13. Alexandrov, N., J. E. Dennis, Jr., R. M. Lewis, V. Torczon. "A Trust Region Framework for Managing the Use of Approximation Models in Optimization". NASA/CR 201745 ICASE Report No 97-50, 1997.
14. Alexandrov, N., and M. Y. Hussaini, editors, *Multidisciplinary Design Optimization: State-of-the-Art*. SIAM, 1997.
15. Asaki, T. "Elasticity-Based TS Warp Cost Functions". Los Alamos National Laboratory Report, 2004.
16. Asaki, T. and M. Sottile. "Improving Parameter Optimization Performance". Los Alamos National Laboratory Report, 2007.
17. Audet, C. and J. E. Dennis, Jr. "Pattern Search Algorithms for Mixed Variable Programming". *SIAM Journal on Optimization*, 11(3):573-594, 2000.
18. Audet, C. and J. E. Dennis, Jr. "Analysis of Generalized Pattern Searches". *SIAM Journal on Optimization*, 13(3):889-903, 2003.
19. Audet, C. and J. E. Dennis, Jr. "A Pattern Search Filter Method for Nonlinear Programming without Derivatives". *SIAM Journal on Optimization*, 14(4):980-1010, 2004.
20. Audet, C. and J. E. Dennis, Jr. "Mesh Adaptive Direct Search Algorithms for Constrained Optimization". *SIAM Journal on Optimization*, 17(2):188-217, 2004.
21. Bandler, J. W., Q. Cheng, S. Dakroury, A. S. Mohamed, M. H. Bakr, K. Madsen, J. Søndergaard. "Space Mapping: The State of The Art". *IEEE Transactions on Microwave Theory and Techniques*, 52(1): 337-361, 2004.
22. Baxter, B. J. C. *The Interpolation Theory of Radial Basis Functions*. Ph.D. thesis, Department of Applied Mathematics and Theoretical Physics, Cambridge University, August 1992.
23. Booker, A. J., J. E. Dennis, Jr, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. "A Rigorous Framework for Optimization of Expensive Functions by Surrogates". *Structural Optimization*, 17(1): 1-13, 1998.
24. Booker, A. J., J. E. Dennis, Jr., P. D. Frank, D. W. Moore, and D. B. Serafini. *Managing Surrogate Objectives to Optimize a Helicopter Rotor Design - Further Experiments*. Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, Missouri, September, 1998.
25. Booker, A. J. "Well-Conditioned Kriging Models for Optimization of Computer Simulations". Technical Report, M&CT-TECH-00-002, The Boeing Company, 2000.

26. Box, G. E. P. and K. B. Wilson. “On the Experimental Attainment of Optimum Conditions”. *Journal of the Royal Statistical Society*, 13(1): 1–45, 1951.
27. Bro-Nielsen, M., C. Gramkow. “Fast Fluid Registration of Medical Images”. *Visualization in Biomedical Computing*. K. H. Hohne and R. Kikinis (eds.). Springer-Verlag, pp. 267–276, 1996.
28. Bro-Nielsen, M., C. Gramkow. *Comparison of Three Filters in the Solution of the Navier-Stokes Equation in Registration*. Proceedings of the Scandinavian Conference on Image Analysis, 795–802, 1997.
29. Buhmann, M. D. “Multivariate Cardinal Interpolation with Radial Basis Functions”. *Constructive Approximations*, 6: 225–256, 1990.
30. Buhmann, M. D. and N. Dyn. “Error Estimates for Multiquadric Interpolation”. *Curves and Surfaces*. P.-J. Laurent, A. Le Méhauté, and L. L. Schumaker (eds.). Academic Press, pp. 51–58, 1991.
31. Buhmann, M. D. *Radial Basis Functions*. Cambridge Monographs on Applied and Computational Mathematics, 2003.
32. Carr, J. C., R. K. Beatson, J. B. Cherrie, T. R. Evans, W. R. Fright, B. C. McCallum, , T. J. McLennan and T. J. Mitchell. “Using RBFs”. <http://www.aranz.com/research/modelling/theory/rbfaq.html>, 2007.
33. Clarke, F. H. *Optimization and Nonsmooth Analysis*. SIAM Classics in Applied Mathematics 5, 2003.
34. Conn, A. R., N. I. M. Gould, and P. L. Toint. “A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds”. *SIAM Journal on Numerical Analysis* 28(2): 545–572, 1991.
35. Coope, I. D. and C. J. Price. “On the Convergence of Grid-Based Methods for Unconstrained Optimization”. *SIAM Journal on Optimization* 11(4): 859–869, 2001.
36. Courant, R., K. Friedrichs, and H. Lewy. “On the Partial Difference Equations of Mathematical Physics”. *IBM Journal* 215–234, 1967, English translation of German original (“Über die Partiellen Differenzgleichungen der Mathematischen Physik”. *Mathematische Annalen*, 100(1): 32–74, 1928).
37. Davis, C. “Theory of Positive Linear Dependence”. *American Journal of Mathematics*, 76(4): 733–746, 1954.
38. Dennis, Jr., J. E. and V. Torczon. “Direct Search Methods on Parallel Machines”. *SIAM Journal on Optimization*, 1:448–474, 1991.
39. Dennis, Jr., J. E. and V. Torczon. “Managing Approximation Models in Optimization”. In Natalia Alexandrov and M. Y. Hussaini, editors, *Multidisciplinary*

- Design Optimization: State-of-the-Art*. SIAM, 1997. Also available as Center for Research on Parallel Computation Report, Rice University, 1995.
40. Dolan, E. D., R. M. Lewis, and V. Torczon. “On the Local Convergence of Pattern Search”. *SIAM Journal on Optimization*, 14(2):567–583, 2003.
 41. Eldred, M. S. and D. M. Dunlavy. *Formulations for Surrogate-Based Optimization with Data Fit, Multifidelity, and Reduced-Order Models*. Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Portsmouth, Virginia, September, 2006.
 42. Fletcher, R. and S. Leyffer. “Nonlinear Programming Without a Penalty Function”. *Mathematical Programming*, 91:239–269, 2002.
 43. Franke, R. “Scattered Data Interpolation: Tests of Some Methods”. *Mathematics of Computation*, 38:181–200, 1982.
 44. Ghia, U., K. N. Ghia, and C. Shin., “High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method”. *Journal of Computational Physics*, 48:387–411, 1982.
 45. Giunta, A. A., M. S. Eldred, and S. F. Wojtkiewicz, Jr. *Overview Of Modern Design of Experiments Methods for Computational Simulations*. Proceedings of the 41st AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, January, 2002.
 46. Griebel, M., T. Dornseifer, and T. Neunhoeffler. *Numerical Simulation in Fluid Dynamics: a Practical Introduction*. SIAM, 1998.
 47. Haber, E. and J. Modersitzki. “Numerical Methods for Volume Preserving Image Registration”. *Inverse Problems*, 20:1621–1638, 2004.
 48. Harlow, F. and J. Shannon. “The Splash of a Liquid Drop”. *Journal of Applied Physics*, 38:3855–3866, 1967.
 49. Hooke, R. and T. A. Jeeves. “Direct Search Solution of Numerical and Statistical Problems”. *Journal of the Association for Computing Machinery*, 8(2):212–229, 1961.
 50. Kokkolaras, M., C. Audet, and J. E. Dennis Jr. “Mixed Variable Optimization of the Number and Composition of Heat Intercepts in a Thermal Insulation System”. *Optimization and Engineering*, 2(1):5–29, 2001.
 51. Lewis, R. M. and V. Torczon. “Rank Ordering and Positive Bases in Pattern Search Algorithms”. Technical Report ICASE 96-71, NASA Langley Research Center, 1996.
 52. Lewis, R. M. and V. Torczon. “Pattern Search Methods for Bound Constrained Minimization”. *SIAM Journal on Optimization*, 9(4):1082–1099, 1999.

53. Lewis, R. M. and V. Torczon. “Pattern Search Methods for Linearly Constrained Minimization”. *SIAM Journal on Optimization*, 10(3):917-941, 2000.
54. Lewis, R. M., V. Torczon, and M.W. Trosset. “Direct Search Methods: Then and Now”. *Journal of Computational and Applied Mathematics*, 124:191-207, 2000.
55. Lewis, R. M. and V. Torczon. “A Globally Convergent Augmented Lagrangian Pattern Search Algorithm for Optimization with General Constraints and Simple Bounds”. *SIAM Journal on Optimization*, 12(4):1075–1089, 2002.
56. Lophaven, S. N., H. B. Nielsen, and J. Søndergaard. “DACE-A MATLAB Kriging Toolbox”. Report IMM-REP-2002-12, Informatics and Mathematical Modelling, Danish Technical University, 2002.
57. Lophaven, S. N., H. B. Nielsen, and J. Søndergaard. “Aspects of the MATLAB Toolbox DACE”. Report IMM-REP-2002-13, Informatics and Mathematical Modelling, Danish Technical University, 2002.
58. Lophaven, S. N., H. B. Nielsen, and J. Søndergaard. “DACE Surrogate Models Software”. <http://www2.imm.dtu.dk/hbn/dace>, 2002.
59. Ma, C. and K.-T. Fang. “A New Approach to Construction of Nearly Uniform Designs”. *International Journal of Materials and Product Technology*, 1(20):115–126, 2004.
60. Magallanez, R. *Surrogate Strategies for Computationally Expensive Optimization Problems with CPU-Time Correlated Functions*. M.S. thesis, Department of Operations Research, Air Force Institute of Technology, March 2007.
61. Martin, J. D. and T. W. Simpson. “Use of Kriging Models to Approximate Deterministic Computer Models”. *American Institute of Aeronautics and Astronautics Journal*, 43(4):853–863, 2005.
62. Modersitzki, J. *Numerical Methods for Image Registration*, Oxford University Press, 2004.
63. Myers, R. H., and D. C. Montgomery. *Response Surface Methodology*. John Wiley and Son, New York, NY, 2002.
64. Nielsen, H. B., S. N. Lophaven, and J. Søndergaard. “Surrogate Modeling by Kriging”. <http://www2.imm.dtu.dk/hbn>, 2003.
65. Nielsen, H. B. “Surrogate Models”. <http://www2.imm.dtu.dk/hbn>, 2004.
66. Nielsen, H. B. and K. F. Thuesen. “Kriging and Radial Basis Functions”. <http://www2.imm.dtu.dk/hbn>, 2005.
67. Notz, W. I. “What Are Computer Experiments and How Do We Design Them”. *Joint Statistical Meetings*, <http://www.stat.ohio-state.edu/compexp>, 2006.

68. O'Reilly, K. R. *Quantitative Object Reconstruction Using Abel Transform Tomography and Mixed Variable Optimization*. M.S. thesis, Department of Operations Research, Air Force Institute of Technology, March 2006.
69. Owen, A. B. "Orthogonal Arrays for Computer Experiments, Integration and Visualization". *Statistica Sinica*, 2:439–452, 1992.
70. Powell, M. J. D. "Univariate Multiquadric Interpolation: Some Recent Results". *Curves and Surfaces*. P. J. Laurent, A. Le Méhauté, and L. L. Schumaker (eds.). Academic Press, pp. 371–381, 1991.
71. Regis, R. and C. Shoemaker. "Parallel Radial Basis Function Methods for the Global Optimization of Expensive Functions". *European Journal of Operational Research*, 182:514–535, 2007.
72. Robinson, T. D., M. S. Eldred, R. Haimes, and K. E. Willcox. *Strategies for Multifidelity Optimization with Variable Dimensional Hierarchical Models*. Proceedings of the 47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference (2nd AIAA Multidisciplinary Design Optimization Specialist Conference), Newport, Rhode Island, May, 2006.
73. Robinson, T. D., M. S. Eldred, R. Haimes, and K. E. Willcox. *Multifidelity Optimization for Variable-Complexity Design*. Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Portsmouth, Virginia, September, 2006.
74. Sacks, J., T. J. Mitchell, W. J. Welch, and H. P. Wynn. "Design and Analysis of Computer Experiments". *Statistical Science*, 4(4):409–435, 1989.
75. Santner, T. J., W. I. Notz, and B. J. Williams. *The Design and Analysis of Computer Experiments*. Springer-Verlag, New York, NY, 2003.
76. Santner, T. "Introductory Overview Lecture on Computer Experiments-The Modeling and Analysis of Data from Computer Experiments". *Joint Statistical Meetings*, <http://www.stat.ohio-state.edu/compexp>, 2006.
77. Schmit, L. A., Jr., and H. Miura. "Approximation Concepts for Efficient Structural Synthesis". Technical Report CR-2552, NASA, 1976.
78. Serafini, D. B. *A Framework for Managing Models in Nonlinear Optimization of Computationally Expensive Functions*. Ph.D. thesis, Department of Computational and Applied Mathematics, Rice University, November 1998.
79. Simpson, T.W., A. J. Booker, D. Ghosh, A. A. Giunta, P. N. Koch, and R. -J. Yang. *Approximation Methods in Multidisciplinary Analysis and Optimization: A Panel Discussion*. 3rd AIAA/ISSMO Internet Conference on Approximations in Optimization, October, 2002.
80. Stein, M. "Large Sample Properties of Simulations Using Latin Hypercube Sampling". *Technometrics*, 29(2):143–151, 1987.

81. Tang, B. “Orthogonal Array-based Latin Hypercubes”. *Journal of the American Statistical Association*, 88(424):1392–1397, 1993.
82. Torczon, V. “On the Convergence of Pattern Search Algorithms”. *SIAM Journal on Optimization*, 7(1):1-25, 1997.
83. Won, K. and T. Ray. “A Framework for Design Optimization Using Surrogates”. *Engineering Optimization*, 37(7):685-703, 2005.

Appendix A. Additional Results and Algorithms

The following Appendix contains additional figures and results for the three test problems referenced throughout the document. The figures of the surrogate response surfaces, for the test problems, give a visual representation of an approximate model of the actual objective function response surface. The plots of the performance history of each test problem illustrate the number of function evaluations required to converge to the optimal solution.

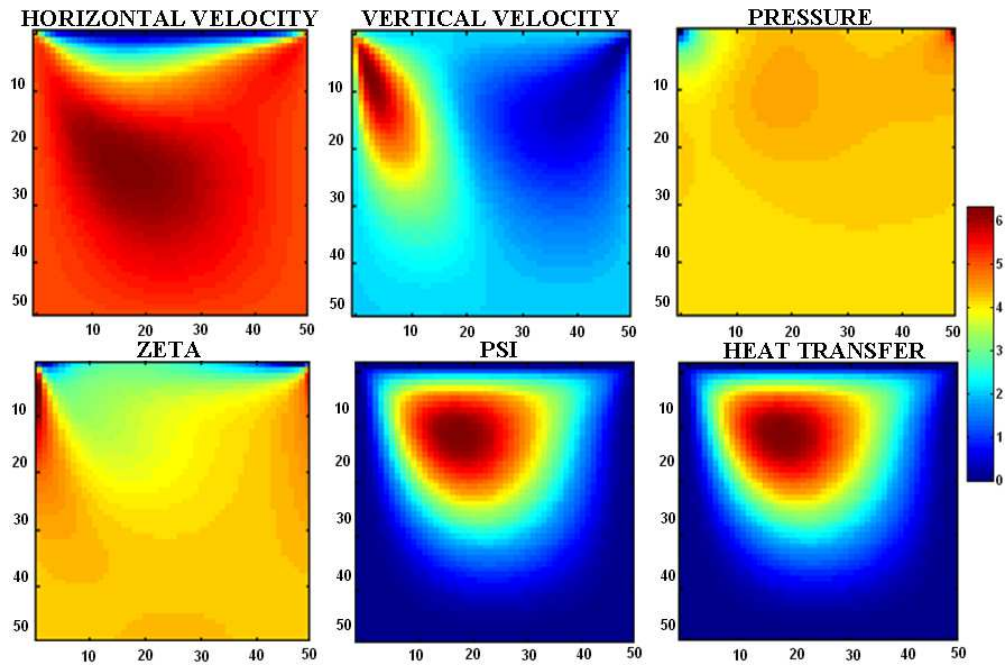


Figure A.1 CFD Simulation Flow Properties of Lid-Driven Cavity

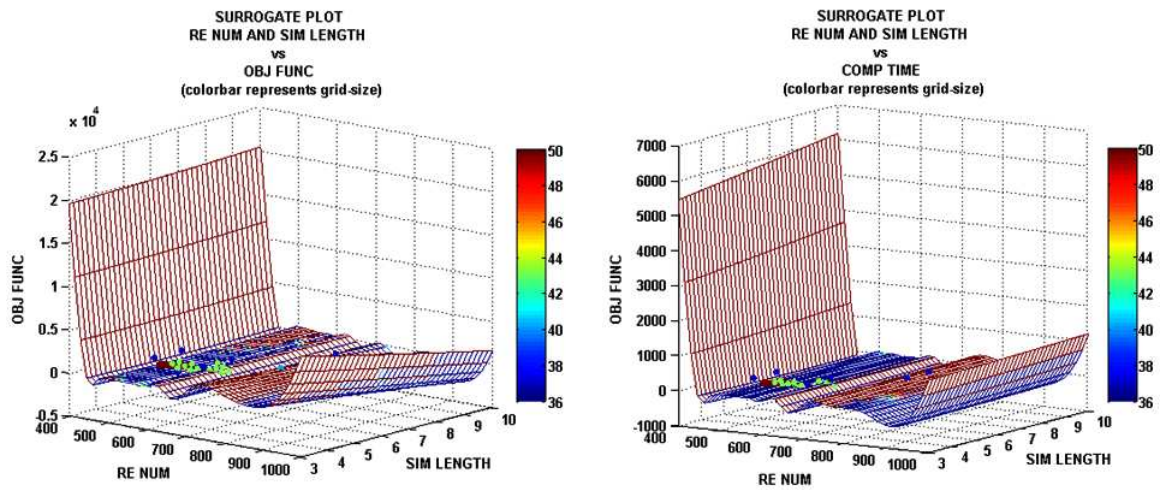


Figure A.2 MVGPS Lid-Driven Cavity Obj. and Time Based Surrogates

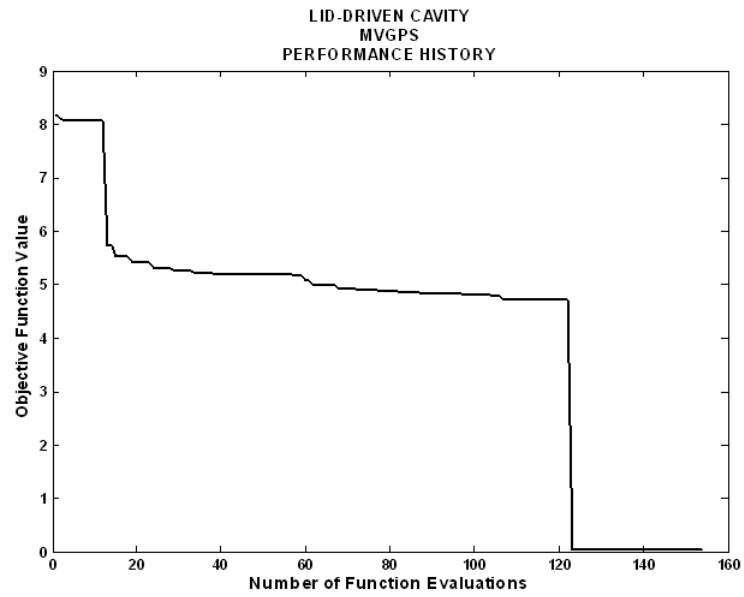


Figure A.3 MVGPS Lid-Driven Cavity Performance History

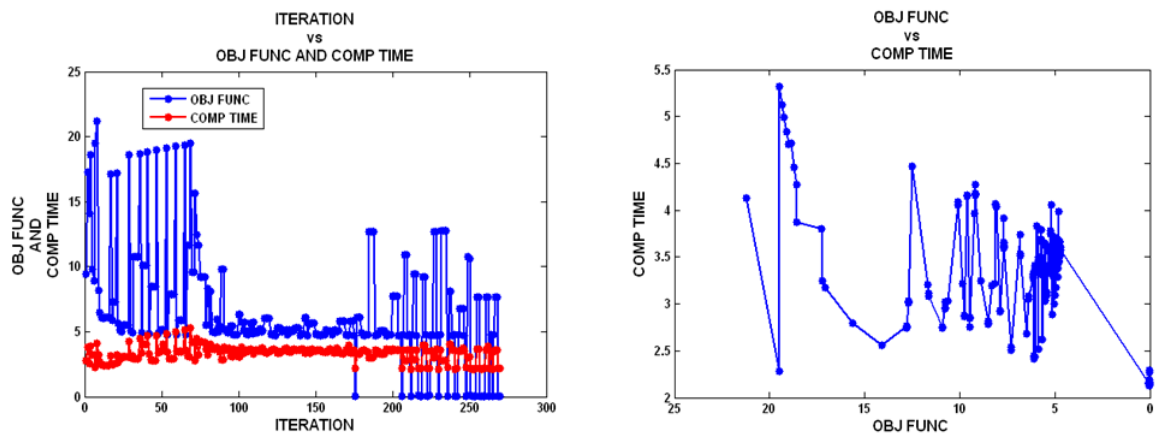


Figure A.4 MVMADS Lid-Driven Cavity Time vs Obj. Function

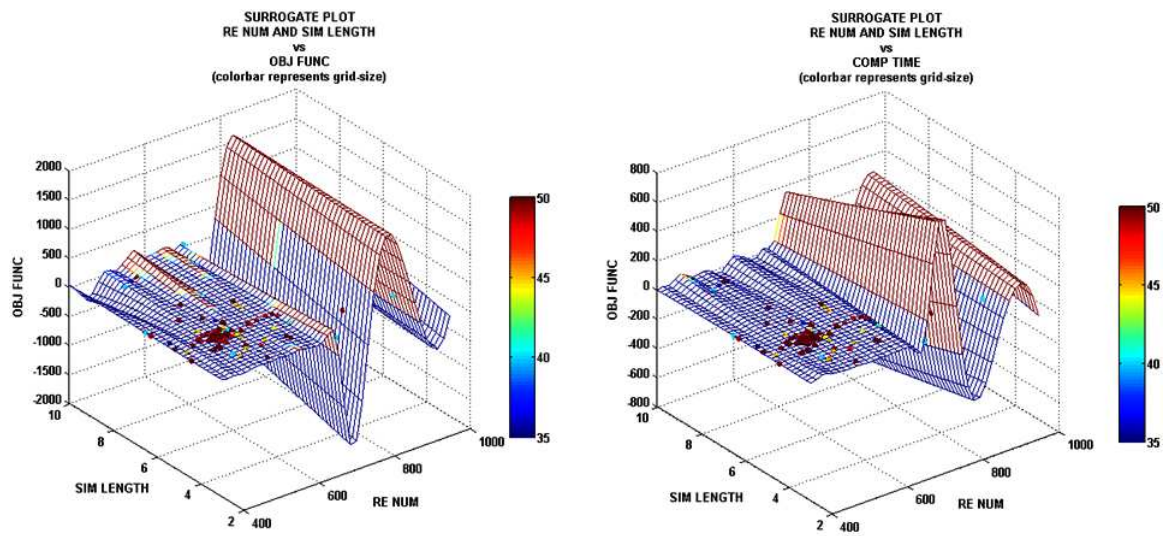


Figure A.5 MVMADS Lid-Driven Cavity Obj. and Time Based Surrogates

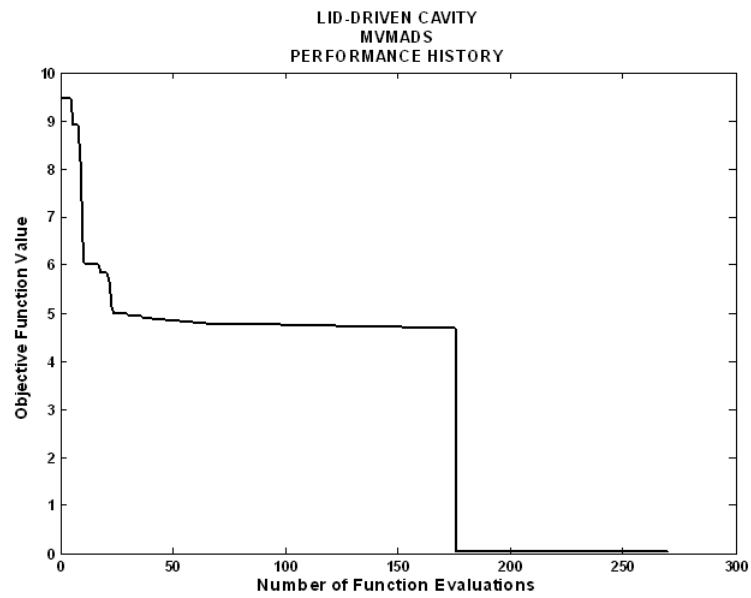


Figure A.6 MVMADS Lid-Driven Cavity Performance History

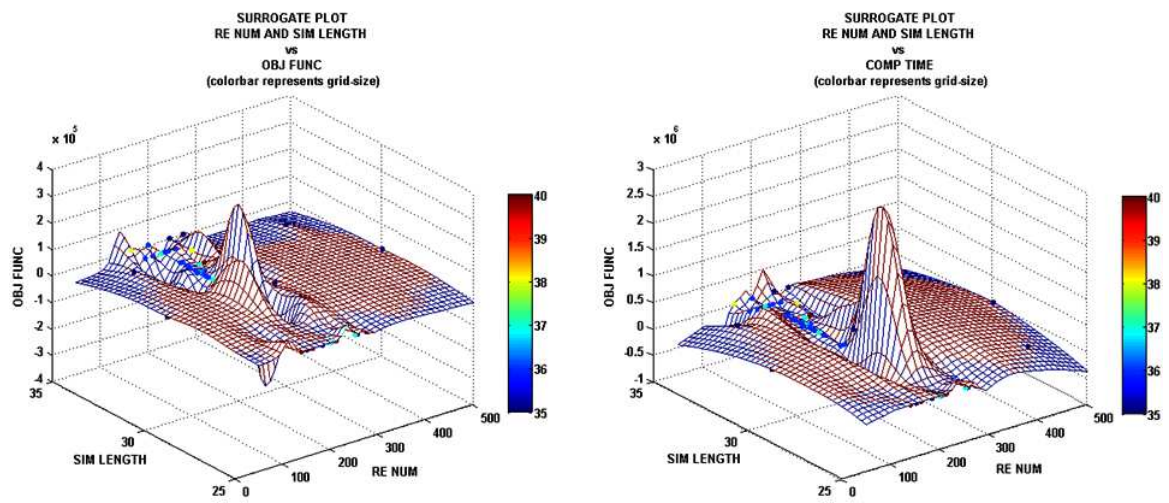


Figure A.7 MVGPS Barrier Flow Obj. and Time Based Surrogates

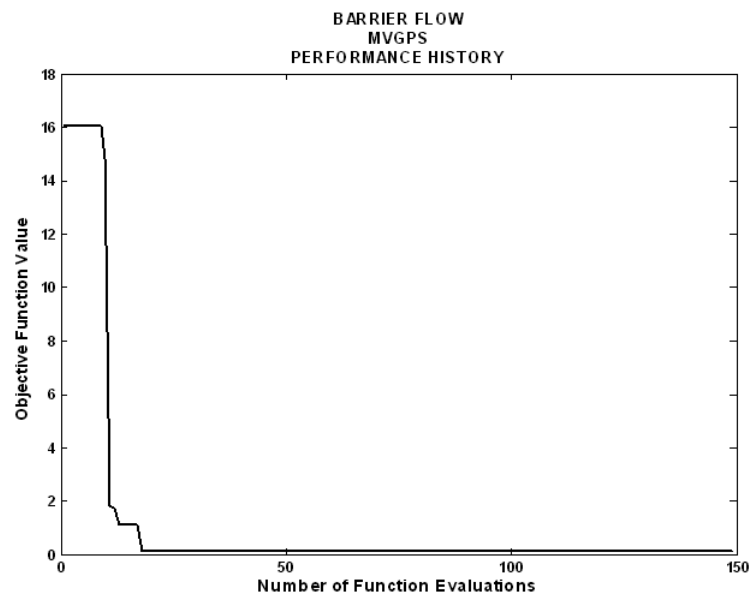


Figure A.8 MVGPS Barrier Flow Performance History

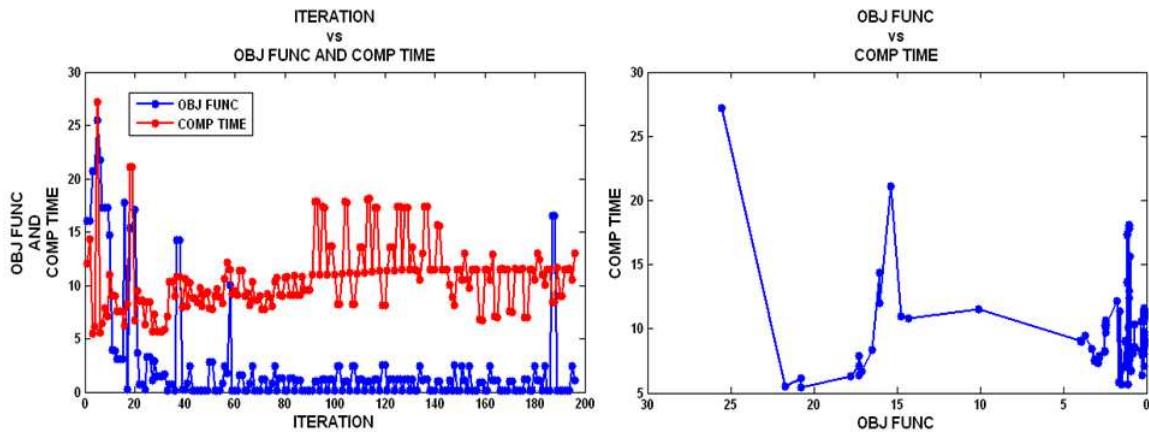


Figure A.9 MVMADS Barrier Flow Time vs Obj. Function

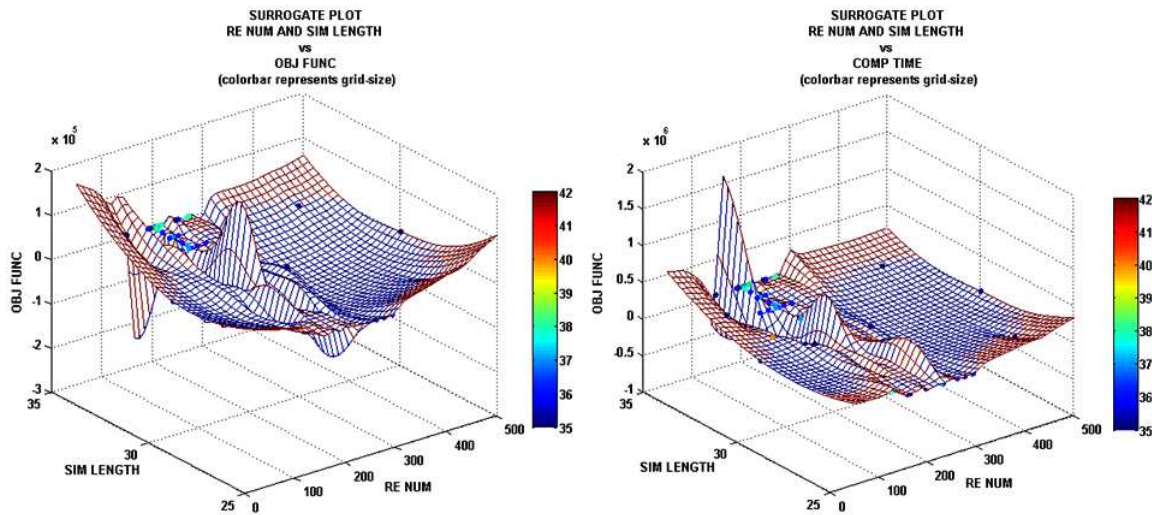


Figure A.10 MVMADS Barrier Flow Obj. and Time Based Surrogates

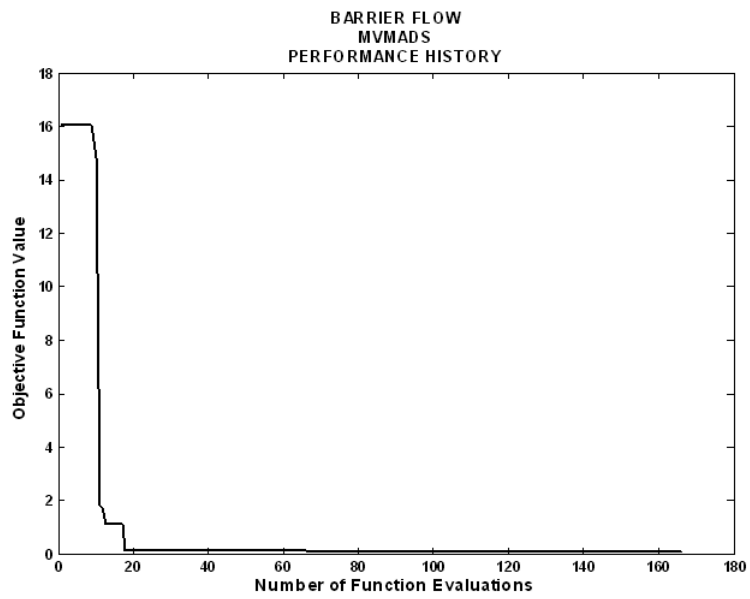


Figure A.11 MVMADS Barrier Flow Performance History

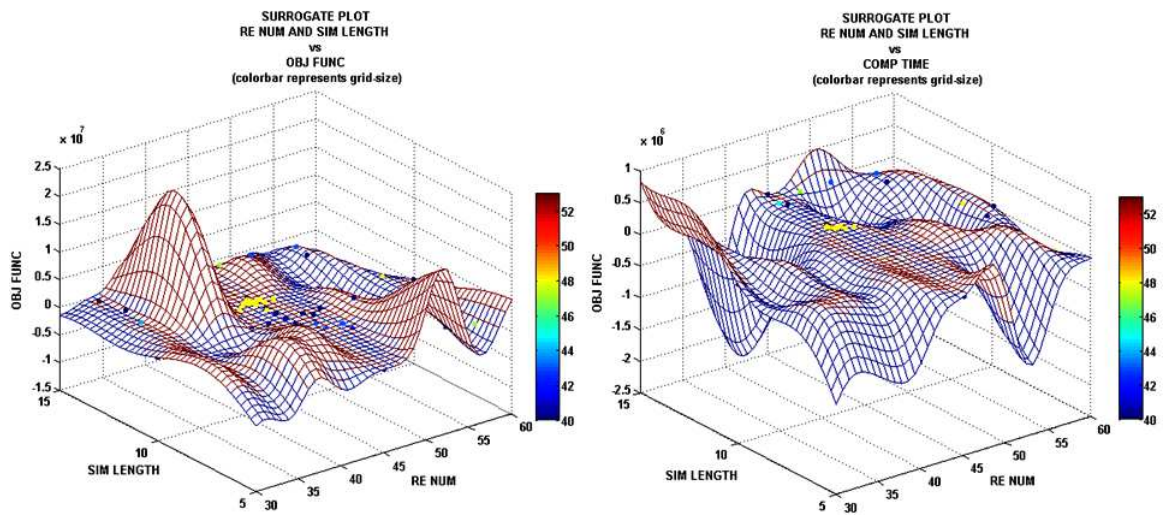


Figure A.12 MVGPS Liquid Drop Obj. and Time Based Surrogates

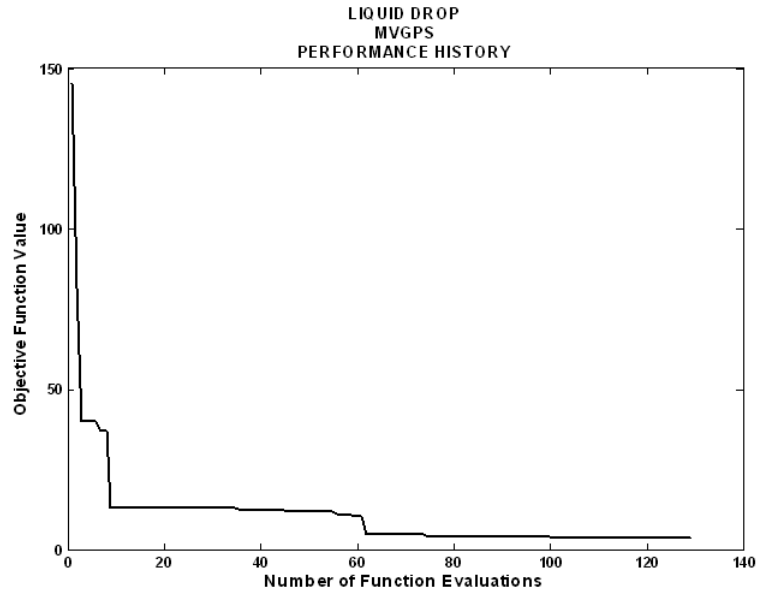


Figure A.13 MVGPS Liquid Drop Performance History

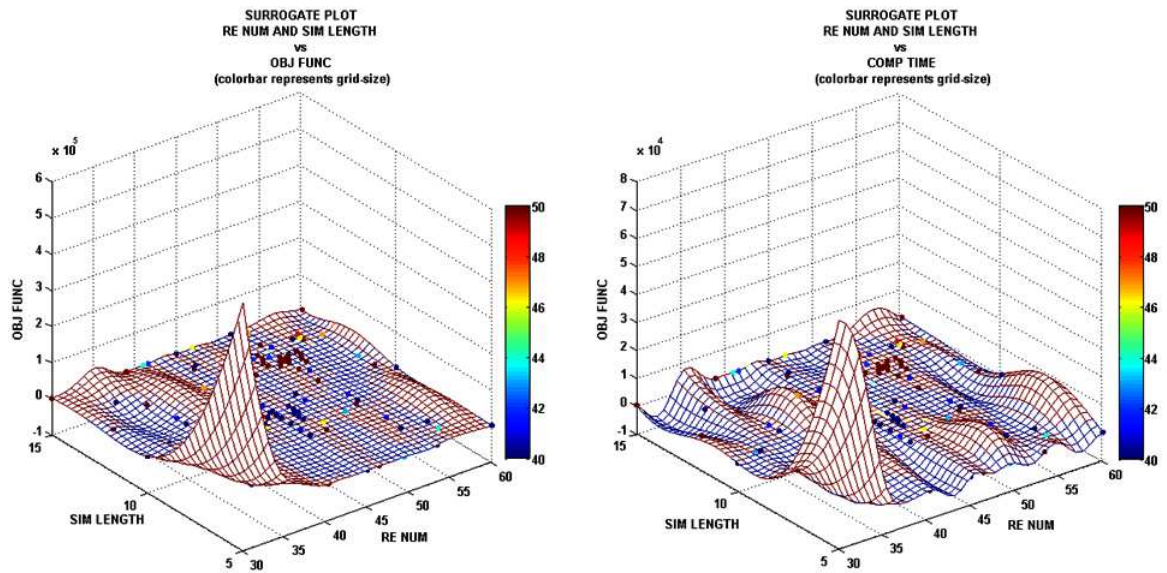


Figure A.14 MVMADS Liquid Drop Obj. and Time Based Surrogates

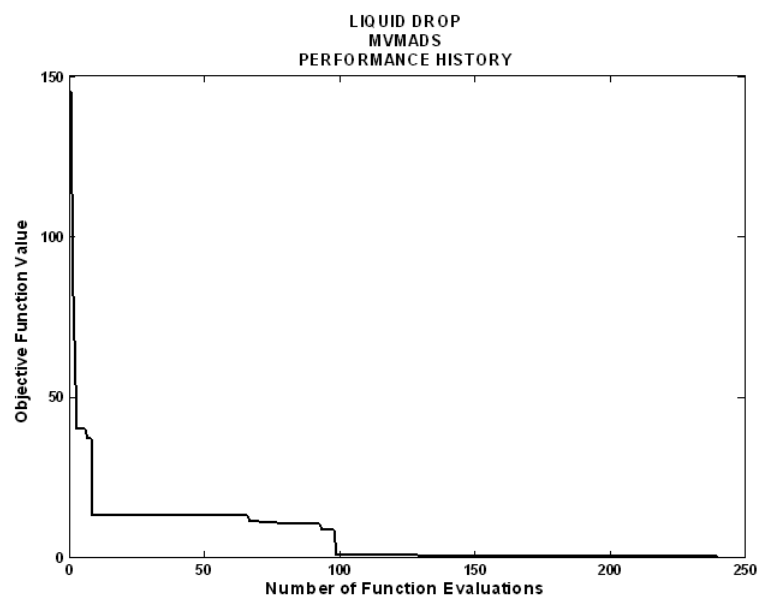


Figure A.15 MVMADS Liquid Drop Performance History

Appendix B. Code for Optimization Framework

The following Appendix contains MATLAB[®] code for executing the TA–MVMADS algorithm 3.4–3.5. The files work in conjunction with the NOMADm [10], DACE [58], and RBF [10] software packages.

```

%*****
function [fx] = cpuproblem(x,p)
%*****
%*****
% CPUPROBLEM IS THE MAIN FUNCTION CALLED BY NOMADM--
% SOLVER FOR NONLINEAR AND MIXED VARIABLE CONSTRAINED OPTIMIZATION WHICH IS
% COPYRIGHT (C) 2001-2007 BY MARK A. ABRAMSON. THIS FUNCTION FILE SETS UP
% THE VARIABLES TO OPTIMIZE ACROSS, CONSTRUCTS THE TRUST REGIONS(IF USED),
% AND ALL PARAMETERS USED FOR SURROGATE CONSTRUCTION. IT RUNS THE
% OPTIMIZATION PROBLEM INVOLVING A DUAL PROCESS OBJECTIVE FUNCTION OF A
% NUMERICAL COMPUTATIONAL FLUID DYNAMICS SIMULATION AND A NUMERICAL IMAGE
% REGISTRATION PROCESS. THE TWO PROCESSES ARE CALLED USING THE COSTFUNC
% FILE

% INPUT VARIABLES:
%   x--VALUE OF THE CONTINUOUS PARAMETERS
%   p--VALUE OF THE DISCRETE OR MIXED VARIABLE
% OUTPUT VARIABLES:
%   fx--OBJECTIVE FUNCTION VALUE RETURNED FOR THE COSTFUNC
%*****
%*****
format long
%*****
% CALLS THE Param DATA THAT IS USED THROUGHOUT THE NOMADm SOFTWARE
Param = getappdata(0, 'PARAM');
%*****
%*****
% DISPLAYS THE MADS ITERATION NUMBER
%*****
display(['MADS ITERATION = ', num2str(Param.madscount)]);
%*****
% SETS THE NUMBER OF FUNCTION EVALUATIONS
%*****
Param.count      = Param.count+1;
Param.data.count = Param.data.count+1;
%*****
%*****
% CHANGING VARIABLES
%*****
pvars.Re      = x(1);
pvars.t_end   = x(2);
pvars.imax    = p{1};
pvars.jmax    = round((1/Param.ratio_pres)*p{1});
%*****
%*****
% CALL TO THE COST FUNCTION WHICH RUNS THE CFD AND THE IMAGE REGISTRATION.
% THE VALUE FX RETURNED IS THE OBJECTIVE FUNCITON VALUE WHICH IS THE
% DIFFERENCE IN THE REFERENCE IMAGE AND THE TRANSFORMED IMAGE. THE VALUE TX
% IS A STRUCTURE THAT CONTAINING THE TIME TO RUN THE CFD AND THE TIME TO
% RUN THE IMAGE REGISTRATION
% *****
[fx,tx] = costfunc(Param.data,Param.tc,pvars);
%*****
%*****
% STORES THE CONTINOUS VARIABLES, DISCRETE VARIABLES, AND OBJECTIVE
% FUNCTION VALUE IN MATRICES FOR USE BY SURROGATES AND POST-PROCESSING
% *NOTE THE VARIABLE DATA.PLOTCOUNT IS ALSO UPDATED WHICH IS ONLY USED FOR
% FILE NAMING OF THE IMAGE REGISTRATION PICS AND THE VARIABLE
% DATA.IMAGEREGPICS IS USED TO DETERMINE WHAT IMAGES TO SAVE; IF
% DATA.IMAGEREGPICS=1 STORE ONLY THE IMAGES OF THE EVALUATIONS RESULTING IN

```

```

% A NEW INCUMBENT. IF DATA.IMAGEREGPICS=2 STORE THE IMAGES OF ALL
% EVALUATIONS
%*****
Param.x(Param.count,:) = x(:);
Param.p(Param.count,:) = p{:};
Param.f(Param.count,:) = fx;
if Param.data.imageregpics==1 && fx < min(Param.fmin)
    Param.data.plotcount = Param.data.plotcount + 1;
elseif Param.data.imageregpics==2
    Param.data.plotcount = Param.data.plotcount + 1;
end
%*****
%*****
% CHECK TO SEE IF THE WARP TIME RETURNED EQUALS -1 WHICH IMPLIES THAT THE
% IMAGE REGISTRATION TIME WAS GREATER THAN THE Param.tc.cutoff VALUE WHICH
% FORMED FROM THE CFD TIME AND WARP TIME ASSOCIATED WITH THE MINIMUM
% OBJECTIVE FUNCTION VALUE SEEN THUS FAR, OR IT IMPLIES NORM OF THE IMAGE
% DIFFERENCES RESULTED IN A DIVERGENT IMAGE DURING THE IMAGE REGISTRATION
% PROCESS.
%*****
if tx.warp < 0
    tx.warp = Inf;
end
%*****
%*****
% CHECK TO SEE IF NON-DIVERGENT VALUES WERE RETURNED FROM THE CFD AND IMAGE
% REGISTRATION. IF THEY WERE, THEN THESE VALUES ARE STORED IN THE
% FOLLOWING PARAMETERS WHICH ARE USED FOR THE SURROGATE CONSTRUCTION. A
% CONSTRUCTION OF TRUST REGION UPPER AND LOWER BOUNDS FOR THE SURROGATE
% FUNCTIONS IS DETERMINED ONLY IF THE USER SPECIFIED THE USE OF A TRUST
% REGION
%*****
if fx ~= Inf
    Param.surcount = Param.surcount+1;
    Param.surxp(Param.surcount,:) = [x(:);p{:}];
    Param.surf(Param.surcount,:) = fx;
    Param.surcf(Param.surcount,:) = tx.cfd;
    Param.surtwarp(Param.surcount,:) = tx.warp;
    Param.surtotal_time(Param.surcount,:) = tx.cfd + tx.warp;
%*****
% CONSTRUCTION OF TRUST REGION UPPER AND LOWER BOUNDS
%*****
if Param.trust_region == 1
    [surf_min,index] = min(Param.surf(1:end));
    Param.surLB = [(Param.surxp(index,1:Param.contdim))'*0.75;...
        floor(Param.surxp(index,Param.totdim)*0.9)];
    Param.surUB = [(Param.surxp(index,1:Param.contdim))'*1.25;...
        ceil(Param.surxp(index,Param.totdim)*1.1)];
    ind_LB = Param.surLB < Param.surLB_orig;
    ind_UB = Param.surUB > Param.surUB_orig;
    Param.surLB(ind_LB) = Param.surLB_orig(ind_LB);
    Param.surUB(ind_UB) = Param.surUB_orig(ind_UB);
%*****
end
end
%*****
%*****
% CHECK THAT ENSURES THE NUMBER OF DESIGN SITES AND RESPONSES USED FOR
% SURROGATE CONSTRUCTION IS LESS THAN OR EQUAL TO THE 3 TIMES THE MINIMUM
% NUMBER OF DATA SITES NEEDED FOR THE HIGHEST ORDER POLYNOMIAL REGRESSION

```

```

% MATRIX. IF MORE THAN 3 TIMES THE NEEDED VALUES EXIST, THE MAXIMUM VALUE
% IS DELETED FROM THE DESIGN SITES AND RESPONSES TO HELP LEAD TO THE
% SOLUTION QUICKER. OF NOTE IS THAT THE TRUST REGION MUST BE TURNED ON OR
% ALL NON-DIVERGENT VALUES ARE USED FOR SURROGATE CONSTRUCTION. THE TRUST
% REGION BOUNDS PREVIOUSLY CREATED ALSO REDUCES THE PARAMETERS USED FOR
% SURROGATE CONSTRUCTION ONLY TO THE VALUES THAT ARE WITHIN THE BOUNDS.
%*****
if (length(Param.surf) >
3*Param.reg_mincol(find(strcmp(func2str(Param.reg_original),Param.reg_choice)))) &&
(Param.trust_region == 1)
    [surf_max,index]          = max(Param.surf(1:end));
    Param.surxp(index,:)      = [];
    Param.surf(index,:)       = [];
    Param.surcfd(index,:)     = [];
    Param.surtwarp(index,:)   = [];
    Param.surtotal_time(index,:) = [];
    %*****
    % REDUCTION OF PARAMETERS USED FOR SURROGATE CONSTRUCTION BASED ON THE
    % SURROGATE UPPER AND LOWER BOUNDS DETERMINED BY THE TRUST REGION UPPER
    % AND LOWER BOUNDS
    %*****
    tr_LB                      = repmat(Param.surLB,length(Param.surf));
    tr_LB(length(Param.surLB)+1:end,:) = [];
    tr_LB                      = tr_LB';
    tr_UB                      = repmat(Param.surUB,length(Param.surf));
    tr_UB(length(Param.surUB)+1:end,:) = [];
    tr_UB                      = tr_UB';
    tr_ind_LB                  = Param.surxp < tr_LB;
    tr_ind_UB                  = Param.surxp > tr_UB;
    [tr_rind_LB,tr_cind_LB]    = find(tr_ind_LB);
    [tr_rind_UB,tr_cind_UB]    = find(tr_ind_UB);
    tr_rind=[tr_rind_LB;tr_rind_UB];
    tr_rind=unique(tr_rind);
    Param.surxp(tr_rind(:),:)  = [];
    Param.surf(tr_rind(:),:)   = [];
    Param.surcfd(tr_rind(:),:) = [];
    Param.surtwarp(tr_rind(:),:) = [];
    Param.surtotal_time(tr_rind(:),:) = [];
    %*****
    % UPDATE THE INDEX OF THE PARAMETERS FOR STORING VALUES FOR SURROGATE
    % CONSTRUCTION
    %*****
    Param.surcount              = length(Param.surf);
end
%*****
%*****
% SET OF TESTS TO DETERMINE IF ENOUGH DATA SITES ARE AVAILABLE FOR THE
% REGRESSION POLYNOMIAL ORDER CHOSEN FOR THE SURROGATES. IF THERE ARE
% ENOUGH DATA SITES THEN THE CHOSEN REGRESSION POLYNOMIAL ORDER IS USED. IF
% THERE ARE NOT ENOUGH THEN IT REDUCES THE POLYNOMIAL ORDER TO THE ORDER
% THAN CAN BE USED GIVEN THE NUMBER OF DATA SITES. THE POLYNOMIAL ORDER
% SPECIFIED BY THE USER IS THE HIGHEST ORDER POLYNOMIAL USED IN THE
% OPTIMIZATION PROCESS. IF A POLYNOMIAL ORDER IS CHOSEN THE SURROGATE WILL
% BE PERFORMED, ELSE THE SURROGATE WILL BE SKIPPED AND THE POLL, NEIGHBOR
% AND POSSIBLY EXTENDED POLL WILL BE EXECUTED
%*****
if Param.count >= Param.initial && length(Param.surf) >= 1
    Param.reg_choice_ind = find(length(Param.surf)>=Param.reg_mincol,1,'last');
    Param.reg = Param.reg_original;
    switch func2str(Param.reg_original)

```

```

        case 'regpoly0'
            if find(strcmp(func2str(Param.reg_original),Param.reg_choice)) <
Param.reg_choice_ind
                Param.reg = Param.reg_original;
            else
                Param.reg = str2func(Param.reg_choice{Param.reg_choice_ind});
            end
        case 'regpoly2reduced'
            if find(strcmp(func2str(Param.reg_original),Param.reg_choice)) <
Param.reg_choice_ind
                Param.reg = Param.reg_original;
            else
                Param.reg = str2func(Param.reg_choice{Param.reg_choice_ind});
            end
        case 'regpoly2'
            if find(strcmp(func2str(Param.reg_original),Param.reg_choice)) <
Param.reg_choice_ind
                Param.reg = Param.reg_original;
            else
                Param.reg = str2func(Param.reg_choice{Param.reg_choice_ind});
            end
        case 'regpoly3reduced'
            if find(strcmp(func2str(Param.reg_original),Param.reg_choice)) <
Param.reg_choice_ind
                Param.reg = Param.reg_original;
            else
                Param.reg = str2func(Param.reg_choice{Param.reg_choice_ind});
            end
        case 'regpoly3'
            if find(strcmp(func2str(Param.reg_original),Param.reg_choice)) <
Param.reg_choice_ind
                Param.reg = Param.reg_original;
            else
                Param.reg = str2func(Param.reg_choice{Param.reg_choice_ind});
            end
    end
    Param.sur_perform = 1;

else
    Param.sur_perform = 0;
end
%*****
%*****
% DETERMINES THE TYPE OF SURROGATE BEING PERFORMED IF THE SURROGATE CHANGER
% IS TURNED ON. THE SURROGATE THAT IS USED DEPENDS ON THE PERCENTAGE SET
% TO THE VARIABLE Param.sur_changer_per. *OF NOTE THE VALUE SET TO
% Param.sur_changer_per IS HOW OFTEN THE "RBF" SURROGATE WILL BE PERFORMED*
%*****
if Param.sur_changer == 1
    if rand >= Param.sur_changer_per
        Param.sur_type = 'dace';
    else
        Param.sur_type = 'rbf';
    end
end
%*****
%*****
% SETS ALL THE VALUES THAT WERE RETURNED FROM THE CFD AND IMAGE
% REGISTRATION RATHER THEY WERE DIVERGENT OR NOT. THE CFD TIME, WARP TIME,
% AND OBJECTIVE FUNCTION VALUES ARE STORED. THE TIME-CUT VALUE IS ALSO SET

```

```

% BASED ON THE CFD AND WARP TIME ASSOCIATED WITH THE MINIMUM OBJECTIVE
% FUNCTION VALUE.
%*****
[f_min,index] = min(Param.f(1:Param.count));

Param.fmin(Param.count,:) = f_min;
Param.data.fmin(Param.count,:) = f_min;
Param.fmax(Param.count,:) = max(Param.f(1:Param.count));

Param.cfd(Param.count,:) = tx.cfd;
Param.cfdmax(Param.count,:) = max(Param.cfd(1:Param.count));
Param.cfdmin(Param.count,:) = min(Param.cfd(1:Param.count));

Param.twarp(Param.count,:) = tx.warp;
Param.twarpmax(Param.count,:) = max(Param.twarp(1:Param.count));
Param.twarpmin(Param.count,:) = min(Param.twarp(1:Param.count));

Param.total_time(Param.count,:) = tx.cfd+tx.warp;
Param.total_timemax(Param.count,:) = max(Param.total_time(1:Param.count));
Param.total_timemin(Param.count,:) = min(Param.total_time(1:Param.count));

Param.warp_cfd_fmin = (Param.twarp(index))+(Param.cfd(index));

if Param.count >= Param.initial
    Param.tc.cutoff = 2*Param.warp_cfd_fmin+.1;
end
%*****
% VALUE NOT USED BY ANY OTHER FUNCTIONS IN THE OPTIMIZATION PROBLEM, BUT
% THE HELP FILE SHOWED USING THIS, SO IT IS SET TO 0 AND NEVER CHANGES.
Param.param = 0;
%*****
% SETS THE Param DATA THAT CHANGED IN THIS FUNCTION AND WILL BE USED
% THROUGHOUT THE NOMADm SOFTWARE
setappdata(0, 'PARAM', Param);
%*****

return

%*****
function Param = cpuproblem_Param
%*****
% DATA INPUT FILE FOR THE COSTFUNC OBJECTIVE FUNCTION FILE FOR THE NOMADM--
% SOLVER FOR NONLINEAR AND MIXED VARIABLE CONSTRAINED OPTIMIZATION WHICH IS
% COPYRIGHT (C) 2001-2007 BY MARK A. ABRAMSON
% THE FOLLOWING FUNCTION INITIALIZES ALL THE PARAM VARIABLES FOR THE
% CPUPROBLEM--MIXED VARIABLE OPTIMIZATION PROBLEM.

% OUTPUT VARIABLE
% Param
% .temp--TEMPORARY PARAMETER FOR STORING THE REFERENCE IMAGE
% .data.refimage--STORES THE REFERENCE IMAGE
% .data.noiselevel--STORES THE NOISE ADDED TO REF IMAGE
% .data.property--STORES THE FLOW PROPERTY ie 'HEAT'
% .data.imagerepics--STORES TYPE OF OPT IMAGES TO SAVE
% .data.prob_name--STORES THE NAME OF THE PROBLEM OPTIMIZING

```

```

% .data.sur_type--STORES THE TYPE OF SURROGATE USED
% .data.count--STORES THE NUMBER OF ITERATES OF OBJ FUNC
% .data.plotcount--STORES THE NUMBER OF IMAGE PLOTS SAVED
% .tc.cfd--TIME VALUE OF CFD ASSOC. WITH MINIMUM OBJ FUNC
% .tc.warp--TIME VALUE OF IMAGE WARP ASSOC WITH MINIMUM OBJ FUNC
% .tc.cutoff--TOTAL TIME ASSOC WITH MINIMUM OBJ FUNC
% .prob_name--STORES NAME OF THE PROBLEM OPTIMIZING
% .LB--LOWER BOUND OF PROBLEM PARAMETER SPACE
% .UB--UPPER BOUND OF PROBLEM PARAMETER SPACE
% .redLB--REDUCED LOWER BOUNDS OF PARAMETER SPACE FOR INITPT(IF USED)
% .redUB--REDUCED UPPER BOUNDS OF PARAMETER SPACE FOR INITPT(IF USED)
% .redBOUNDS--DETERMINES IF REDUCED BOUNDS ARE USED FOR INITPT
% .discreteLB--LOWER BOUND OF THE DISCRETE PARAMETER SPACE
% .discreteUB--UPPER BOUND OF THE DISCRETE PARAMETER SPACE
% .discrete_init--INITIAL VALUE OF DISCRETE PARAMETER
% .surLB--LOWER BOUND OF SURROGATE PROBLEM
% .surUB--UPPER BOUND OF SURROGATE PROBLEM
% .ratio_pres--ratio that preserves the spatial discretization of CFD
% .data.imagereg_iters--MAXIMUM ITERS FOR THE IMAGE WARP CODE
% .surLB_orig--ORIGINAL LOWER BOUND OF SURROGATE PROBLEM(IF TR USED)
% .surUB_orig--ORIGINAL UPPER BOUND OF SURROGATE PROBLEM(IF TR USED)
% .DOEtype--TYPE OF DOE TO USE FOR THE INITIAL POINTS
% .CCDtype--IF CCD USED FOR INITIAL POINTS, TYPE OF CCD TO USE
% .initial--NUMBER OF INITIAL POINTS USED
% .count--NUMBER OF FUNCTION EVALUATION
% .madscount--NUMBER OF MADS ITERATIONS
% .surcount--NUMBER OF SURROGATE IMPLEMENTATIONS
% .f--OBJ FUNCTION VALUE
% .x--VALUE OF CONTINUOUS PARAMETERS
% .p--VALUE OF DISCRETE PARAMETERS
% .surxp--STORES THE DISC AND CONT PARAMETERS FOR SURROGATE CONSTRUCT
% .surf--STORES THE OBJ FUNC VALUE FOR SURROGATE CONSTRUCT
% .surcfd--STORES THE CFD TIME FOR SURROGATE CONSTRUCT
% .surtwarp--STORES THE WARP TIME FOR SURROGATE CONSTRUCT
% .surtotal_time--STORES TOTAL TIME FOR SURROGATE CONSTRUCT
% .fmin--STORES THE MINIMUM OBJ FUNC VALUE
% .fmax--STORES THE MAXIMUM OBJ FUNC VALUE
% .cfd--STORES THE CFD TIME OF OBJ FUNC EVALUATION
% .cfdmax--STORES THE MAXIMUM CFD TIME OF OBJ FUNC EVALUATION
% .cfdmin--STORES THE MINIMUM CFD TIME OF OBJ FUNC EVALUATION
% .twarp--STORES THE WARP TIME OF THE OBJ FUNC EVALUATION
% .twarpmax--STORES THE MAXIMUM WARP TIME OF THE OBJ FUNC EVALUATION
% .twarpmin--STORES THE MINIMUM WARP TIME OF THE OBJ FUNC EVALUATION
% .total_time--STORES THE TOTAL TIME OF THE OBJ FUNC EVALUATION
% .total_timemax--STORES THE MAX TOTAL TIME OF THE OBJ FUNC EVAL
% .total_timemin--STORES THE MIN TOTAL TIME OF THE OBJ FUNC EVAL
% .warp_cfd_fmin--STORES THE WARP AND CFD TIME ASSOC WITH MIN OBJ VAL
% .tc.cutoff--2 TIMES THE WARP AND CFD TIME ASSOC WITH MIN OBJ VALUE
% .sur_info--STORES ALL SURROGATE CONSTRUCTION INFO AT EACH SUR BUILD
% .no_sur_info--STORES SAME INFO AS ABOVE FOR EACH NO SUR BUILD
% .sursearch--STORES THE NUMBER OF SURROGATES IMPLEMENTED
% .surnosearch--STORES THE NUMBER OF NO SURROGATE IMPLEMENTATIONS
% .surstopsearch--USER SPECIFIED TIME SO THAT NO SURROGATES ARE BUILT
% .sur_perform--DETERMINES IF A BUILT SURROGATE IS TO BE SOLVED
% .condnum_thresh--THRESHOLD FOR THE CONDITION NUMBER OF SUR BUILD
% .condnum--CONDITION NUMBER OF THE SYSTEM ASSOC WITH SUR BUILD
% .contdim--DIMENSION OF THE CONTINUOUS VARIABLES
% .discdim--DIMENSION OF THE DISCRETE VARIABLES
% .totaldim--TOTAL DIMENSION OF PARAMETER SPACE(CONT AND DISC)
% .param--VARIABLE THAT IS SET TO ZERO

```

```

% .cpu_time--STORES THE OVERALL COMPUTATION TIME OF OPTIM PROCESS
% .sur_type--TYPE OF SURROGATE USED DACE OR RBF
% .rbfkernel--KERNEL TYPE FOR RBF(IF USED)
% .sur_changer--DETERMINES IF OPT PROCESS CAN CHANGE SURROGATE TYPES
% .sur_changer_per--PERCENT OF TIME TO CHANGE FROM DACE TO RBF
% .dace_surbuilder_count--NUMBER OF DACE SURROGATES BUILT
% .rbf_surbuilder_count--NUMBER OF RBF SURROGATES BUILT
% .surbuilder_count--NUMBER OF SURROGATES BUILT
% .reg--ORDER OF REGRESSION MODEL USED FOR DACE AND RBF
% .reg_original--ORIGINAL ORDER OF REGRESSION MODEL(CAN STEP DOWN)
% .corr--CORRELATION MODEL USED FOR DACE
% .trust_region--DETERMINES IF TRUST REGIONS ARE USED
% .theta_init--INITIAL VALUE OF THETA OPTIMIZED BY DACE
% .opt_theta--OPTIMAL THETA VALUE RETURNED BY DACE
% .reg_mincol--MINIMUM NUMBER OF DESIGN SITES REQ FOR SURR POLY MODEL
% .reg_choice--NAME OF ALL POSSIBLE REG MODELS USED BY DACE AND RBF
% .reg_choice_ind--INDEX OF THE NAME OF THE REG MODEL USED
%*****
%*****
% LOAD A REFERENCE IMAGE. WE ARE TRYING TO RECOVER THE SIMULATION PARAMTERS
% THAT PRODUCED THIS IMAGE
%*****
Param.temp = struct2cell(load('refimage03.mat'));
%*****
%*****
% INVERT BECAUSE THE 'HEAT' MAPS ARE OF NEGATIVE VALUES AND WE LIKE
% POSITIVE VALUED ARRAYS
%*****
Param.data.refimage = Param.temp{1};
%*****
%*****
% SET NOISE LEVEL - NOTE THAT THE CONSTANT FACTOR MUST MATCH THE ACTUAL
% NOISE OF THE REFERENCE IMAGE.
%*****
temp2 = struct2cell(load('NOISE.mat'));
Param.data.noiselevel = norm(temp2{1},'fro');
%*****
%*****
% THIS PROPERTY SHOULD NOT BE CHANGED. IT SPECIFIES THAT WE ARE CONSIDERING
% THE HEAT DISTRIBUTION IMAGE OUTPUT FROM THE SIMULATION (SEE COSTFUNC.M)
%*****
Param.data.property = 'U';
%*****
%*****
% VARIABLES USED TO SAVE THE IMAGE REGISTRATION PICTURES **ENTER A "0" FOR
% NO PICTURES; ENTER A "1" TO SAVE THE PICTURES OF ONLY THE INCUMBENTS**;
% ENTER A 2 TO SAVE PICTURES OF ALL EVALUATIONS. THE PROBLEM NAME,
% SURROGATE TYPE, COUNT, AND PLOT COUNT ARE USED FOR NAMING THE IMAGE
% REGISTRATION PICTURES. **VALID VALUES FOR THE PROBLEM NAME ARE:
% lid-driven cavity, convection, drop and splash, flow past obstacle**
% **VALID VALUES FOR sur_type are: none, dace, rbf, dace-rbf**
%*****
Param.data.imageregpics = 0;
Param.data.prob_name = 'flow past obstacle';
Param.data.sur_type = 'dace';
Param.data.count = 0;
Param.data.plotcount = 0;
%*****
%*****
% INITIAL TIME-CUT VALUES FOR THE CFD, IMAGE REGISTRATION, AND TOTAL TIME

```



```

% CUTOFF PARAMETERS FOR THE COSTFUNC AND CWAFFIT FILE THAT RUN THE CFD AND
% THE IMAGE REGISTRATION
%*****
Param.tc.cfd=Inf;
Param.tc.warp=Inf;
Param.tc.cutoff=Inf;
%*****
% INPUT FILENAME AND SET UP THE BOUNDS, PRESERVING RATION OF SPATIAL
% DISCRETIZATION, AND THE MAX NUMBER OF ITERS FOR THE IMAGE WARPING
%*****
Param.prob_name = 'flow past obstacle'; %CHANGE TO APPROPRIATE PROB NAME

switch upper(Param.prob_name)
case 'LID-DRIVEN CAVITY'
    Param.discreteLB = 35;
    Param.discreteUB = 50;
    Param.discrete_init = 40;
    Param.ratio_pres = 1;
    Param.LB = [400; 3];
    Param.UB = [1000; 10];
    Param.surLB = [400; 3; Param.discreteLB];
    Param.surUB = [1000; 10; Param.discreteUB];
    Param.data.imagereg_iters = 200;
case 'CONVECTION'
    Param.discreteLB = 5;
    Param.discreteUB = 25;
    Param.discrete_init = 5;
    Param.ratio_pres = 1;
    Param.LB = [500; 3000];
    Param.UB = [1500; 6000];
    Param.surLB = [500; 3000; Param.discreteLB];
    Param.surUB = [1500; 6000; Param.discreteUB];
    Param.data.imagereg_iters = 200;
case 'DROP AND SPLASH'
    Param.discreteLB = 35;
    Param.discreteUB = 50;
    Param.discrete_init = 40;
    Param.ratio_pres = 1;
    Param.LB = [30; 5];
    Param.UB = [60; 15];
    Param.surLB = [30; 5; Param.discreteLB];
    Param.surUB = [60; 15; Param.discreteUB];
    Param.data.imagereg_iters = 200;
case 'FLOW PAST OBSTACLE'
    Param.discreteLB = 35;
    Param.discreteUB = 50;
    Param.discrete_init = 40;
    Param.ratio_pres = 1;
    Param.LB = [50; 25];
    Param.UB = [500; 35];
    Param.surLB = [50; 25; Param.discreteLB];
    Param.surUB = [500; 35; Param.discreteUB];
    Param.data.imagereg_iters = 200;
end

Param.redBOUNDS = 0; %0--NO REDUCED BOUNDS; 1--REDUCED BOUNDS

Param.redLB = Param.LB;

```

```

Param.redUB = Param.UB;

Param.surLB_orig = Param.surLB;
Param.surUB_orig = Param.surUB;
%*****
%*****
% SET UP THE TYPE OF DOE TO USE AND THE TYPE OF CCD(IF USED)
%*****
Param.DOEtype = 'CCD'; %CCD; LHS; UNICUT; UNIDESIGN; CENTER
Param.CCDtype = 'inscribed'; %INSCRIBED; CIRCUMSCRIBED; FACED
%*****
%*****
% INITIALIZATION OF ALL VARIABLES USED THROUGHOUT OPTIMIZATION PROCESS
%*****
Param.initial      = 0;
Param.count        = 0;
Param.madscount    = 0;
Param.surcount     = 0;
Param.f            = [];
Param.x            = [];
Param.p            = [];
Param.surxp        = [];
Param.surf         = [];
Param.surcfd       = [];
Param.surtwarp     = [];
Param.surttotal_time = [];

Param.fmin = [Inf];
Param.fmax = [];

Param.cfd      = [];
Param.cfdmax   = [];
Param.cfdmin   = [];

Param.twarp    = [];
Param.twarpmax = [];
Param.twarpmin = [];

Param.total_time      = [];
Param.total_timemax  = [];
Param.total_timemin  = [];

Param.warp_cfd_fmin = [];

Param.sur_info      = {};
Param.no_sur_info   = {};
Param.sursearch     = 0;
Param.surnosearch   = 0;
Param.surstopsearch = 0; % SET TO VALUE GREATER THAN ZERO IF DESIRED
Param.sur_perform    = 0;
Param.condnum_thresh = 1/eps; % SET TO LOWER VALUE IF DESIRED
Param.condnum        = 0;
%*****
%*****
% SET UP DIMENSION OF PROBLEM
%*****
Param.contdim = 2; % CHANGE TO APPROPRIATE NUMBER OF CONT VARIABLES
Param.discdim = 1; % CHANGE TO APPROPRIATE NUMBER OF DISC VARIABLES

```

```

Param.totaldim = Param.contdim + Param.discdim;
%*****
%*****
% INITIALIZE THE OVERALL CPUTIME AND THE UNUSED VARIABLE .PARAM
%*****
Param.param          = 0;
Param.cpu_time      = 0;
%*****
%*****
% SET UP THE SURROGATE PARAMETERS AND THE TRUST REGION PARAMETERS; CERTAIN
% SURROGATE COUNTING VARIABLES ARE INITIALIZED
%*****
Param.sur_type       = 'dace'; % CHANGE TO 'none', 'dace', 'rbf',
                             % 'dace-rbf'

Param.rbfkernel      = 'multiquadric'; % CHANGE TO 'bi-harmonic',
                                     % 'tri-harmonic',
                                     % 'gaussian',
                                     % 'multiquadric',
                                     % 'inv-multiquadric',
                                     % 'thinplatespline'

Param.sur_changer    = 0; % 0--NO CHANGE FROM DACE TO RBF;
                       % 1--CHANGE FROM DACE TO RBF;

Param.sur_changer_per = 0; % PERCENT OF TIME TO CHANGE FROM DACE
                           % TO RBF--VALUE 0 TO 1

Param.rbf_surbuilder_count = 0;
Param.dace_surbuilder_count = 0;
Param.surbuilder_count     = 0;

Param.reg             = @regpoly2; % CHANGE TO @regpoly0,
                                   % @regpoly2reduced, @regpoly2,
                                   % @regpoly3reduced, @regpoly3

Param.reg_original    = @regpoly2; % MUST EQUAL SAME VALUE AS
                                   % Param.reg

Param.corr            = @corrgauss; % CHANGE TO @correxp,
                                   % @correxpg, @corrgauss
                                   % @corrlin, @corrspherical,
                                   % @corrspline, @corrcubic

    Param.trust_region = 1; % 0--NO TRUST REGION; 1--TRUST REGION
    Param.theta_init   = 10*ones(1,Param.totaldim);
Param.opt_theta       = 10*ones(1,Param.totaldim);
%*****
%*****
% SETS UP THE MINIMUM NUMBER OF COLUMNS REQUIRED FOR A USER SPECIFIED
% REGRESSION MODEL ORDER; INITIALIZES ALL VALUES USED IN OPTIMIZATION
% PROCESS THAT ALL THE REGRESSION ORDER TO MOVE TO A LOWER ORDER POLYNOMIAL
% IF NECESSARY
%*****
Param.reg_mincol = [1;(1+2*Param.totaldim);((Param.totaldim+1)*
(Param.totaldim+2)/2);(1+3*Param.totaldim);((Param.totaldim+1)*
(Param.totaldim+2)/2)+Param.totaldim];

```

```

Param.reg_choice = {'regpoly0','regpoly2reduced','regpoly2',
                   'regpoly3reduced','regpoly3'};
Param.reg_choice_ind = 0;
%*****
%*****
% SETS THE Param DATA THAT CHANGED IN THIS FUNCTION AND WILL BE USED
% THROUGHOUT THE NOMADm SOFTWARE
%*****
setappdata(0,'PARAM',Param);
%*****
return

%*****
function [A,l,u,plist] = cpuproblem_Omega(n,p)
%*****
%*****
% CPUPROBLEM_OMEGA SETS UP THE DOMAIN OF THE MAIN OPTIMIZATION PROBLEM

% INPUT:
%   n--DIMENSIONALITY OF THE PROBLEM
%   p--VALUE OF THE DISCRETE PARAMETERS
% OUTPUT:
%   A--IDENTITY MATRIX FOR SETTING UP THE BOUND CONSTRAINTS
%   l--LOWER BOUND OF THE CONTINUOUS PARAMETERS
%   u--UPPER BOUND OF THE CONTINUOUS PARAMETERS
%   plist--BOUNDS OF THE DISCRETE VARIABLES
%*****
%*****
% CALLS THE Param DATA THAT IS USED THROUGHOUT THE NOMADm SOFTWARE
Param = getappdata(0,'PARAM');
%*****
%*****
% SET UP THE CONSTRAINTS AND THE BOUNDS
%*****
A = eye(n);
l = Param.LB;
u = Param.UB;
plist{1}={Param.discreteLB:Param.discreteUB};
%*****
%*****
return

%*****
function iterate = cpuproblem_x0
%*****
%*****
% CPUPROBLEM_XO DETERMINES THE SET OF INITIAL POINTS BASED ON A DESIGN OF
% EXPERIMENT AS SPECIFIED BY THE USER. CHOICES OF DESIGNS ARE
% CCD(INSCRIBED, CIRCUMSCRIBED, OR FACE-CENTERED), LATIN HYPERCUBE, UNIFORM
% DESIGN, OR A CENTER POINT.

% OUTPUT:
%   iterate--THE SET OF INITIAL ITERATES EVALUATED BY THE OBJECTIVE
%   FUNCTION FOR DEVELOPING THE INITIAL SURROGATE FUNCTION
%*****
%*****
% CALLS THE Param DATA THAT IS USED THROUGHOUT THE NOMADm SOFTWARE
%*****

```

```

Param = getappdata(0, 'PARAM');
%*****
%*****
% DETERMINE IF REDUCED BOUNDS ARE SPECIFIED BY THE USER. THIS ALLOWS THE
% USER TO REDUCE THE DOMAIN THE DESIGN OF EXPERIMENTS DETERMINES POINTS
%*****
if Param.redBOUNDS == 1
    Param.redLB = ((Param.UB-Param.LB)/8)+Param.LB;
    Param.redUB = (((Param.UB-Param.LB)/8)*7)+Param.LB;
end
%*****
%*****
% DETERMINE THE TYPE OF DESIGN OF EXPERIMENT USED TO FIND THE SET OF
% INITIAL POINTS
%*****
switch upper(Param.DOEtype)
    %*****
    % A CCD DESIGN IS USED. DETERMINE THE TYPE OF CCD: INSCRIBED,
    % CIRCUMSCRIBED, OR FACE CENTERED. USE THE CCD DESIGN TO FIND THE SET
    % OF CODED INITIAL POINTS [-1,1]
    case 'CCD'
        DOE = ccdesign(Param.contdim, 'center', 1, 'fraction', 0, 'type',
            lower(Param.CCDtype));
        a = size(DOE,1);
        %*****
        % UNCODE THE SET OF INITIAL POINTS [-1,1] TO THE TRUE DESIGN
        % VARIABLES OVER THE TRUE PARAMETER SPACE. THE DISCRETE VARIABLE IS
        % HELD CONSTANT FOR THE INITIAL POINTS AND SPECIFIED BY THE USER
        switch upper(Param.CCDtype)
            case 'INSCRIBED'
                for i = 1:a
                    for j = 1:Param.contdim
                        dsite = DOE(i,j);
                        if dsite == 1
                            iterate(i).x(j,1) = Param.redUB(j,1);
                            iterate(i).p = {Param.discrete_init};
                        elseif dsite == -1
                            iterate(i).x(j,1) = Param.redLB(j,1);
                            iterate(i).p = {Param.discrete_init};
                        elseif dsite == 0
                            iterate(i).x(j,1) = (Param.redUB(j,1) +
                                Param.redLB(j,1))/2;
                            iterate(i).p = {Param.discrete_init};
                        elseif (dsite == -(sqrt(Param.contdim)/Param.contdim)
                            || dsite == (sqrt(Param.contdim)/Param.contdim))
                            iterate(i).x(j,1) = ((Param.redUB(j,1) +
                                Param.redLB(j,1))/2) + (((Param.redUB(j,1) +
                                Param.redLB(j,1))/2) - Param.redLB(j,1))*dsite);
                            iterate(i).p = {Param.discrete_init};
                        end
                    end
                end
            case 'CIRCUMSCRIBED'
                for i = 1:a
                    for j = 1:Param.contdim
                        dsite = DOE(i,j);
                        if dsite == 1
                            dsite = sqrt(Param.contdim)/Param.contdim;
                            iterate(i).x(j,1) = ((Param.redUB(j,1) +
                                Param.redLB(j,1))/2) + (((Param.redUB(j,1) +

```

```

        Param.redLB(j,1))/2)-Param.redLB(j,1))*dsite);
        iterate(i).p      = {Param.discrete_init};
elseif dsite == -1
        dsite = -sqrt(Param.contdim)/Param.contdim;
        iterate(i).x(j,1) = ((Param.redUB(j,1)+
        Param.redLB(j,1))/2)+(((Param.redUB(j,1)+
        Param.redLB(j,1))/2)-Param.redLB(j,1))*dsite);
        iterate(i).p      = {Param.discrete_init};
elseif dsite == 0
        iterate(i).x(j,1) = (Param.redUB(j,1)+
        Param.redLB(j,1))/2;
        iterate(i).p      = {Param.discrete_init};
elseif dsite/Param.contdim > 0
        iterate(i).x(j,1) = Param.redUB(j,1);
        iterate(i).p      = {Param.discrete_init};
elseif dsite/Param.contdim < 0
        iterate(i).x(j,1) = Param.redLB(j,1);
        iterate(i).p      = {Param.discrete_init};
end
end
end
case 'FACED'
for i = 1:a
for j = 1:Param.contdim
dsite = DOE(i,j);
if dsite == 1
        iterate(i).x(j,1) = Param.redUB(j,1);
        iterate(i).p      = {Param.discrete_init};
elseif dsite == -1
        iterate(i).x(j,1) = Param.redLB(j,1);
        iterate(i).p      = {Param.discrete_init};
elseif dsite == 0
        iterate(i).x(j,1) = (Param.redUB(j,1)+
        Param.redLB(j,1))/2;
        iterate(i).p      = {Param.discrete_init};
end
end
end
end
end
%*****
% A LHS DESIGN IS USED TO FIND THE CODED SET OF INITIAL POINTS [0,1]
case 'LHS'
rand('state',0)
DOE=
lhsdesign(Param.totaldim^2,Param.contdim,'iterations',5,'criterion',
'maximin','criterion','correlation');
a = size(DOE,1);
%*****
% UNCODE THE SET OF INITIAL POINTS [-1,1] TO THE TRUE DESIGN
% VARIABLES OVER THE TRUE PARAMETER SPACE. THE DISCRETE VARIABLE IS
% HELD CONSTANT FOR THE INITIAL POINTS AND SPECIFIED BY THE USER
for i = 1:a
for j = 1:Param.contdim
dsite = DOE(i,j);
iterate(i).x(j,1) = ((Param.redUB(j,1)-
Param.redLB(j,1))*dsite)+Param.redLB(j,1);
iterate(i).p      = {Param.discrete_init};
end
end
end
%*****

```

```

% A UNIFORM DESIGN IS USED TO FIND THE CODED SET OF INITIAL POINTS [0,1]
case 'UNICUT'
    rand('state',0)
    DOE = uniformcut(Param.contdim,Param.totaldim^2);
    a = size(DOE,1);
    %*****
    % UNCODE THE SET OF INITIAL POINTS [0,1] TO THE TRUE DESIGN
    % VARIABLES OVER THE TRUE PARAMETER SPACE. THE DISCRETE VARIABLE IS
    % HELD CONSTANT FOR THE INITIAL POINTS AND SPECIFIED BY THE USER
    for i = 1:a
        for j = 1:Param.contdim
            dsite = DOE(i,j);
            iterate(i).x(j,1) = ((Param.redUB(j,1)-
                Param.redLB(j,1))*dsite)+Param.redLB(j,1);
            iterate(i).p = {Param.discrete_init};
        end
    end
    %*****
    % A UNIFORM DESIGN IS USED. USE THE LHS DESIGN TO FIND THE SET
    % OF CODED INITIAL POINTS [0,1]
    case 'UNIDESIGN'
        rand('state',0)
        DOE = uniformdesign(Param.contdim,Param.totaldim^2);
        a = size(DOE,1);
        %*****
        % UNCODE THE SET OF INITIAL POINTS [0,1] TO THE TRUE DESIGN
        % VARIABLES OVER THE TRUE PARAMETER SPACE. THE DISCRETE VARIABLE IS
        % HELD CONSTANT FOR THE INITIAL POINTS AND SPECIFIED BY THE USER
        for i = 1:a
            for j = 1:Param.contdim
                dsite = DOE(i,j);
                iterate(i).x(j,1) = ((Param.redUB(j,1)-
                    Param.redLB(j,1))*dsite)+Param.redLB(j,1);
                iterate(i).p = {Param.discrete_init};
            end
        end
    endend
    end
    %*****
    % CENTER POINT DESIGN
    case 'CENTER'
        iterate.x = (Param.LB + Param.UB)/2;
        iterate.p = {Param.discrete_init};
    end
    %*****
    % DETERMINE THE NUMBER OF INITIAL POINTS BEING USED BECAUSE THE TIME CUT
    % PARAMETER IS NOT IMPLEMENTED DURING THE INITIAL POINTS EVALUATION BY THE
    % OBJECTIVE FUNCTION
    Param.initial=length([iterate.x]);
    %*****
    % SETS THE Param DATA THAT CHANGED IN THIS FUNCTION AND WILL BE USED
    % THROUGHOUT THE NOMADm SOFTWARE
    setappdata(0,'PARAM',Param);
    %*****
    return

```

```

%*****
function S = searchsurrogate(Problem,Options,Search,delta,pcenter)
%*****
%*****
% SEARCHSURROGATE IS FUNCTION USED FOR BUILDING AND SOLVING THE FUNCTIONAL
% SURROGATE OPTIMIZATION PROBLEM. IT ALSO BUILDS A COMPUTATIONAL TIME
% SURROGATE FOR ORDERING THE FEASIBLE VALUES THAT ARE RETURNED FROM THE
% FUNCTIONAL SURROGATE FOR EVALUATION IN THE OBJECTIVE FUNCTION

% INPUT
% Problem--STRUCTURE THAT CONTAINS ALL THE FILE NAMES ASSOCIATED WITH THE
% SURROGATE OPTIMIZATION PROBLEM
% Options--OPTIONS OF THE MADS SOFTWARE FOR POLL TYPE, TERMINATION
% CRITERIA, ETC FOR THE OPTIMIZATION OF THE SURROGATE PROBLEM
% Search--SETS UP ALL OF THE SEARCH OPTIONS; NUMBER OF PTS RETURNED,
% TERMINATION CRITERIA, ETC
% delta--THE MESH SIZE AT TERMINATION
% pcenter--THE CURRENT BEST FEASIBLE SOLUTION
% OUTPUT
% S--THE POINTS RETURNED FROM THE SURROGATE OPTIMIZATION AND ORDERED BY
% THE TIME BASED SURROGATE TO BE EVALUATED IN THE OBJECTIVE FUNCTION
%*****
%*****
% CALLS THE Param DATA THAT IS USED THROUGHOUT THE NOMADm SOFTWARE
%*****
Param = getappdata(0,'PARAM');
%*****
%*****
% SET UP THE PROBLEM, DEFAULTS AND OPTIONS FOR THE SURROGATE OPTIMIZATION
% PROBLEM: PROBLEM FILE NAMES, OPTIONS, TERMINATION CRITERIA USED BY THE
% MADS SOFTWARE FOR OPTIMIZING THE SURROGATE
%*****
Defaults = mads_defaults('Surrogate'); % MADS SURROGATE DEFAULTS
Options = Defaults.Options; % MADS DEFAULT OPTIONS

Problem.nameCache = 'SCACHE'; % CACHE NAME FOR SURROGATE
Problem.typeProblem = 'STRUTH'; % SURROGATE PROBLEM TYPE

Problem.File.F = 'searchsurrogate_eval'; % FUNCTIONS FILE
Problem.File.O = 'searchsurrogate_Omega'; % LINEAR CONSTRAINTS FILE
Problem.File.X = 'searchsurrogate_X'; % CLOSED CONSTRAINTS FILE
Problem.File.I = 'searchsurrogate_x0'; % INITIAL POINTS FILE
Problem.File.N = 'searchsurrogate_N'; % DISCRETE NEIGHBOR FILE
(MVP ONLY)

Problem.File.P = 'searchsurrogate_Param'; % PARAMETER FILE
Problem.File.C = 'searchsurrogate_Cache.mat'; % PREVIOUSLY CREATED CACHE
FILE

Problem.File.S = 'searchsurrogate_Session.mat'; % PREVIOUSLY CREATED
SESSION FILE

Problem.File.H = 'searchsurrogate_History.txt'; % ITERATION HISTORY TEXT
FILE

Problem.File.D = 'searchsurrogate_Debug.txt'; % DEBUG LOG FILE
Problem.fType = 'M'; % TYPE OF FUNCTIONS FILE
{M,F,C}

Problem.nc = 0; % NUMBER OF NONLINEAR
CONSTRAINTS

Problem.iterate0 = {}; % SETS ALL THE ITERATE VALUES TO EMPTY SO THAT
% MULTIPLE VALUES CAN BE RETURNED FOR EVALUATION

```



```

% FROM THE SURROGATE OPTIMIZATION IN THE OBJECTIVE
% FUNCTION
%*****
% SPECIFY SEARCH OPTIONS FOR THE SURROGATE OPTIMIZATION PROBLEM
Options.nSearches      = 0; % SPECIFY THE NUMBER OF SEARCHES=0
Search.nPoints        = 4; % SPECIFY THE NUMBER OF POINTS TO RETURN
                        % FROM THE SURROGATE OPTIMIZATION PROBLEM
                        % FOR EVALUATION IN THE OBJ FUNC

Options.Poll.type      = 'Standard_2n'; % SPECIFY THE TYPE OF POLL STEP
                        % FOR THE SURROGATE PROBLEM;
                        % 'Standard_2n' or MADS_2n' OR
                        % SEE MADS DEFAULTS FOR OTHER
                        % CHOICES

Options.Poll.order     = 'Consecutive'; % FOR CHOICES, SEE MADS_DEFAULTS
Options.Poll.center    = 0; % POLL AROUND N-TH FILTER POINT
Options.Poll.complete  = 0; % FLAG FOR COMPLETE POLLING
Options.EPoll.completeN = 0; % FLAG FOR COMPLETE NEIGHBOR POLLING
Options.EPoll.complete = 0; % FLAG FOR COMPLETE EXTENDED POLLING
%*****
% SPECIFY SEARCH TERMINATION CRITERIA
Options.Term.delta     = 1e-4; % MINIMUM MESH SIZE
Options.Term.nIter     = Inf; % MAXIMUM NUMBER OF ITERATIONS
Options.Term.nFunc     = 50000; % MAXIMUM NUMBER OF FUNCTION EVALS
Options.Term.time      = Inf; % MAXIMUM CPU TIME
Options.Term.nFails    = Inf; % MAX NUMBER OF CONSECUTIVE POLL FAILS
%*****
% SPECIFY MESH CONTROL CRITERIA
Options.delta0         = 1.0; % INITIAL MESH SIZE
Options.deltaMax       = Inf; % BOUND ON HOW COARSE THE MESH CAN GET
Options.meshRefine     = 0.5; % MESH REFINEMENT FACTOR
Options.meshCoarsen    = 1.0; % MESH COARSENING FACTOR
%*****
% SPECIFY CHOICES FOR FILTER MANAGEMENT (FOR PROBLEMS WITH NONLINEAR
% CONSTRATINS
Options.hmin           = 1e-8; % MINIMUM INFEASIBLE POINT H-VALUE
Options.hmax           = 1.0; % MAXIMUM H-VALUE OF A FILTER POINT
%*****
% SPECIFY CHOICES FOR THE EXTENDED POLL TRIGGER (FOR MIXED VARIABLE PROBLEMS
Options.EPoll.fTrigger = 0.01; % F-VALUE EXTENDED POLL TRIGGER
%*****
% SPECIFY CHOICES FOR MADS FLAG PARAMETERS: CACHE, HISTORY, PLOTS, ETC
Options.loadCache      = 1; % LOAD PRE-EXISTING CACHE FILE
Options.countCache     = 1; % COUNT CACHE POINTS AS FUNCTION CALLS
Options.scale          = 2; % SCALE DIRECTIONS USING THIS LOG BASE
Options.degeneracyScheme = 'random'; % SCHEME FOR DEGENERATE CONSTRAINTS
Options.saveHistory    = 0; % SAVES MADS PERFORMANCE TO TEXT FILE
Options.plotHistory    = 0; % PLOT MADS PERFORMANCE
Options.plotFilter     = 0; % PLOT THE FILTER REAL-TIME
Options.plotColor      = 'k'; % COLOR OF HISTORY PLOT
Options.debug          = 3; % TURN ON STATUS MESSAGES FOR DEBUGGING
Options.Sur.Term.delta = 0.01; % SURROGATE MINIMUM MESH SIZE
%*****
%*****
% CALL TO SEARCHSURROGATE PARAM FILE USED TO BUILD THE DACE OR RBF
% SURROGATES
[opt_theta,min_cond] = searchsurrogate_Param;
%*****
% CALLS THE Param DATA THAT IS USED THROUGHOUT THE NOMADm SOFTWARE
Param = getappdata(0, 'PARAM');

```

```

%*****
% STORES THE OPTIMAL THETA VALUES RETURNED FROM A DACE SURROGATE BUILD,
% INITIALIZES THETA_INIT FOR THE NEXT DACE SURROGATE BUILD, AND THE
% CONDITION NUMBER OF THE CORRELATION MODEL OF THE DACE SURROGATE
Param.opt_theta=[Param.opt_theta;opt_theta];
Param.theta_init=10*ones(1,Param.totaldim);
Param.condnum=min_cond;
%*****
% SETS THE Param DATA THAT CHANGED IN THIS FUNCTION AND WILL BE USED
% THROUGHOUT THE NOMADm SOFTWARE
setappdata(0,'PARAM',Param);
%*****
% CHECK FOR THE TIME CUTOFF VALUE BEING LARGER THAN THE USER SPECIFIED TIME
% THRESHOLD, THE CONDITION NUMBER OF THE CORRELATION MODEL BEING LARGER
% THAN THE CONDITION NUMBER THRESHOLD, AND IF THE SURROGATE PERFORM FLAG IS
% TURNED ON
%*****
if (Param.tc.cutoff > Param.surstopsearch) && (Param.condnum <
    Param.condnum_thresh) && (Param.sur_perform == 1)
    %*****
    % UPDATE THE SURROGATE COUNTER, MADS ITERATION COUNTER, AND SETS THE
    % Param DATA THAT CHANGED IN THIS FUNCTION AND WILL BE USED THROUGHOUT
    % THE NOMADm SOFTWARE
    Param.sursearch = Param.sursearch+1;
    Param.madscount = Param.madscount + 1;
    setappdata(0,'PARAM',Param);
    %*****
    % STORE THE SURROGATE INFO SETS THE Param DATA THAT CHANGED IN THIS
    % FUNCTION AND WILL BE USED THROUGHOUT THE NOMADm SOFTWARE

    Param.sur_info(Param.sursearch,:)= {num2str(Param.sursearch),
    num2str(Param.madscount),Param.sur_type,func2str(Param.reg)};
    setappdata(0,'PARAM',Param);
    %*****
    % DISPLAY THE MADS COUNT AND SURROGATE INFO ON THE SCREEN
    display(['MADS ITERATION = ',num2str(Param.madscount)])
    display([upper(Param.sur_type), ' ',upper(func2str(Param.reg)),' ',...
        'SEARCH SURROGATE = ',num2str(Param.sursearch)]);
    %*****
    % INITIALIZE THE ITERATE TO BEGIN THE SURROGATE OPTIMIZATION PROBLEM TO
    % BE THE CONTINUOUS VARIABLES AUGMENTED WITH THE DISCRETE VARIABLES
    % BECAUSE THE SURROGATE CAN ONLY OPTIMIZE ON CONTINUOUS VALUES
    iterate0.x = [pcenter.x ; pcenter.p{:}];
    iterate0.p = {};
    %*****
    % CALL THE MADS SOFTWARE TO SOLVE THE SURROGATE OPTIMIZATION PROBLEM
    [BestF,BestI,RunStats,SurCache] = mads(Problem,iterate0,Options);
    %*****
    % RETRIEVE AND EVALUATE THE NUMBER OF POINTS SPECIFIED BY THE USER TO
    % BE RETURNED FROM THE SURROGATE OPTIMIZATION
    indFeasible = 1:min(Search.nPoints,length(SurCache.Filter.feasible));
    tempS = SurCache.iterate(SurCache.Filter.feasible(indFeasible));
    %*****
    % CREATE A TEMP VECTOR FOR STORING THE BEST 4 ITERATES OF THE OPTIMIZED
    % SURROGATE FOR FUNCTION VALUE PREDICTOR AND CHANGE THE CONTINUOUS
    % VALUES ASSOCIATED WITH THE DISCRETE VARIABLES TO THE FLOOR AND
    % CEILING OF THOSE VALUES
    tempS = [tempS.x]';
    %*****
    % IF THE TEMP VECTOR IS EMPTY RETURN THE ORIGINAL POLL CENTER AS THE

```

```

% POINT TO BE EVALUATED BY THE OBJECTIVE FUNCTION
if isempty(tempS)
    S.x = pcenter.x;
    S.p = pcenter.p;
%*****
% IF THE TEMP VECTOR IS NOT EMPTY CONVERT THE POSSIBLE CONTINUOUS VALUE
% ASSOCIATED WITH THE DISCRETE VARIABLE BACK TO DISCRETE VALUES USING
% THE FLOOR AND CEILING
else
    tempS = [tempS(:,1:end-1), floor(tempS(:,end)); tempS(:,1:end-
1), ceil(tempS(:,end))];
%*****
% EVALUATE THE RETURNED VALUES IN THE TIME BASED SURROGATE FOR
% ORDERING IN ASCENDING COMPUTATIONAL TIME AND EVALUATION IN THE
% OBJECTIVE FUNCTION(CALLS AND SETS THE Param DATA USED IN THIS
% FUNCTION AND THROUGHOUT THE OPTIMIZATION
switch upper(Param.sur_type)
    case 'DACE'
        Param = getappdata(0, 'PARAM');
        [SUR_Tx] = predictor(tempS, Param.dmodelT);
        setappdata(0, 'PARAM', Param);
    case 'RBF'
        Param = getappdata(0, 'PARAM');
        for k = 1:size(tempS,1)
            [SUR_Tx(k)] = evalRBF(tempS(k, :)', Param.rbfmodelT);
        end
        setappdata(0, 'PARAM', Param);
end
%*****
% SORT THE VALUES EVALUATED BY THE TIME BASED SURROGATE IN
% ASCENDING ORDER OF COMPUTATIONAL TIME AND USE THE INDICES OF THE
% SORT TO ORDER THE VALUES OF tempS TO BE RETURNED FOR EVALUATION
% BY THE OBJECTIVE FUNCTION
[tempSUR_Tx, ind] = sort(SUR_Tx);
tempS = tempS(ind, :);
%*****
% STORES THE SORTED VALUES BACK INTO THE STRUCTURE FORMAT USED BY
% MADS. ONLY HALF THE VALUES ARE RETURNED BECAUSE THE FLOOR AND
% CEILING FUNCTION CAN POSSIBLY DOUBLE THE ACTUAL NUMBER OF VALUES
% SPECIFIED BY THE USER FOR EVALUATION
for k = 1:(length(tempS)/2)
    S(k).x = tempS(1:end-1, k);
    S(k).p = {tempS(end, k)};
end
end
%*****
*****
F THE SURROGATE IS NOT PERFORMED THE NO SEARCH COUNTER IS UPDATED, MADS
OUNT IS UPDATED, THE NO SEARCH INFO IS UPDATED AND THE VALUE RETURNED
OR EVALUATION IS THE ORIGINAL POLL CENTER
*****
=
Param.surnosearch=Param.surnosearch+1;
Param.madscount = Param.madscount + 1;

Param.no_sur_info(Param.surnosearch, :)=
{num2str(Param.surnosearch), num2str(Param.madscount),
Param.sur_type, func2str(Param.reg)};

display(['MADS ITERATION = ', num2str(Param.madscount)])

```

```

display([upper(Param.sur_type), ' ', upper(func2str(Param.reg)), ' ', ...
        'NO SEARCH SURROGATE = ', num2str(Param.surnosearch)]);

S.x = pcenter.x;
S.p = pcenter.p;
end
%*****
%*****
% SETS THE Param DATA THAT CHANGED IN THIS FUNCTION AND WILL BE USED
% THROUGHOUT THE NOMADm SOFTWARE
setappdata(0, 'PARAM', Param);
%*****
%*****
return

%*****
function [opt_theta,min_cond] = searchsurrogate_Param
%*****
%*****
% SEARCHSURROGATE_PARAM BUILDS THE DACE OR RBF SURROGATE. IF A DACE
% SURROGATE IS USED UPPER AND LOWER BOUNDS ARE CALCULATED FOR THE THETA
% VALUE BEFORE THE DACE SURROGATE IS BUILT

% OUTPUT
% opt_theta--THE OPTIMIZED THETA VALUE RETURNED FROM A DACE SURROGATE
% BUILD
% min_cond--THE CONDITION NUMBER OF THE CORRELATION MODEL OR THE BASIS
% FUNCTION RETURNED TO THE SEARCHSURROGATE TO DETERMINE IF THE THRESHOLD
% TEST IS MET AND IF THE SEARCH STEP WILL USE THE SURROGATE OR NOT
%*****
%*****
% CALLS THE Param DATA THAT IS USED THROUGHOUT THE NOMADm SOFTWARE
%*****
Param = getappdata(0, 'PARAM');
%*****
%*****
% DETERMINE THE TYPE OF SURROGATE USED. IF A DACE SURROGATE IS USED THE
% UPPER AND LOWER BOUNDS FOR THETA ARE OPTIMIZED BEFORE THE DACE SURROGATE
% IS BUILT
%*****
if strcmp(upper(Param.sur_type), 'DACE') %&& Param.sursearch == 1
%*****
% UPDATE THE SURROGATE BUILD COUNTER AND THE DACE SURROGATE BUILDER
% COUNT
Param.surbuilder_count = Param.surbuilder_count + 1;
Param.dace_surbuilder_count = Param.dace_surbuilder_count + 1;
%*****
% INITIALIZE THE CONDITION NUMBER OF THE CORRELATION MODEL R TO ZERO
% BEFORE THE UPPER AND LOWER BOUNDS ON THETA ARE CALCULATED
condR = 0;
%*****
% COMPUTE THE LOWER BOUND ON THETA. WHILE THE CONDITION NUMBER OF R IS
% NOT BAD AND THE LOWER BOUND OF THETA IS GREATER THAN OR EQUAL TO 0.01
% CONTINUE BUILD DACE MODELS AND DIVIDE THE INITIAL VALUE OF THETA BY
% TWO EACH TIME
while condR <= 1e8 && all(Param.theta_init >= 0.01)
[dmodel] = dacefit(Param.surxp, Param.surf, Param.reg, Param.corr,
Param.theta_init);
condR = condest(dmodel.R);
end

```

```

        Param.theta_init = Param.theta_init/2;
end
%*****
% MULTIPLY THE FINAL VALUE OF THETA BY TWO JUST IN CASE THE VALUE IS
% LESS THAN 0.01
theta0    = 2*Param.theta_init;
%*****
% SET THE LOWER BOUND VALUE
Param.lob = theta0;
%*****
%*****
% INITIALIZE THE VALUE OF THE NORM OF THE GRADIENT TO INF BEFORE
% OPTIMIZING THE UPPER BOUND
dLnorm = inf;
h       = 0.1;
%*****
% COMPUTE THE UPPER BOUND ON THETA. WHILE NORM OF THE GRADIENT BETWEEN
% TWO DACE CORRELATION MODELS IS GREATER THAN 0.01 CONTINUE BUILDING
% TWO DACE MODELS AND OPTIMIZING THETA AND MULTIPLYING THE THETA LOWER
% BOUND VALUE BY 2 EACH TIME
while dLnorm > 1e-2
    theta1 = theta0+h*ones(1,length(theta0));
    dmodel0 = dacefit(Param.surxp,Param.surf,Param.reg, Param.corr,
    theta0);
    dmodel1 = dacefit(Param.surxp,Param.surf,Param.reg, Param.corr,
    theta1);
    G = dmodel0.gamma';
    S = dmodel0.sigma2;
    %*****
    % IF SVD WAS NOT USED FOR OPTIMIZING THETA THE GRADIENT NORM IS
    % COMPUTED USING THE CHOLESKY FACTOR
    if isnan(dmodel0.E)
        C0 = dmodel0.C';
        R0 = C0*C0';
        R1 = dmodel1.C*dmodel1.C';
        dR = (R1 - R0)./h;
        dL = (G'*dR*G)/(2*S)-trace(C0'\(C0\dR));
    %*****
    % IF SVD WAS USED FOR OPTIMIZING THETA THE GRADIENT NORM IS
    % COMPUTED USING CORRELATION MODEL
    else
        R0 = dmodel0.R;
        R1 = dmodel1.R;
        dR = (R1 - R0)./h;
        dL = (G'*dR*G)/(2*S)-trace(dmodel0.E*dR);
    end
    dLnorm = norm(dL);
    theta0 = 2*theta0;
end
%*****
% DIVIDE THE FINAL VALUE OF THE UPPER BOUND ON THETA BY TWO TO ENSURE
% THE UPPER BOUND IS NOT TO LARGE
Param.upb = theta0/2;
%*****
% SET THE INITIAL VALUE OF THETA TO THE MIDPOINT OF THE OPTIMIZED UPPER
% AND LOWER BOUNDS AS THE INITIAL GUESS FOR THE THETA VALUE USED IN
% BUILDING THE FUNCTION AND TIME BASED DACE SURROGATES
Param.theta = (Param.lob+Param.upb)./2;
%*****
%*****

```

```

% BUILD THE FUNCTION AND TIME BASED DACE SURROGATES
[Param.dmodelF] =
dacefit(Param.surxp,Param.surf,Param.reg,Param.corr,Param.theta,
Param.lob,Param.upb);
[Param.dmodelT] =
dacefit(Param.surxp,Param.surtotal_time,Param.reg,Param.corr,
Param.theta,Param.lob,Param.upb);
%*****
% COMPUTE THE OPTIMAL THETA VALUE AND THE CONDITION NUMBER OF THE
% CORRELATION MODELS FOR USE IN THE OPTIMIZATION PROCESS FOR
% DETERMINING IF THE CONDITION NUMBER THRESHOLD IS MET AND SETTING THE
% NEXT VALUE OF THETA TO USE DURING A DACE SURROGATE BUILD
opt_theta = Param.dmodelF.theta;
condRF = condest(Param.dmodelF.R);
condRT = condest(Param.dmodelT.R);
min_cond = min(condRF,condRT);
%*****
% DISPLAY THE SURROGATE BUILD INFORMATION
display(['SURROGATE BUILD = ',num2str(Param.surbuilder_count),...
' (' ,upper(Param.sur_type), ' ',upper(func2str(Param.reg)),')'])
display([upper(Param.sur_type), ' SURROGATES BUILT = ',...
num2str(Param.dace_surbuilder_count), ' : THETA = ',...
mat2str(opt_theta), ' ', 'CONDITION NUMBER =
',num2str(min_cond,'%10.8e')]);
%*****
% DETERMINE IF AN RBF SURROGATE IS TO BUILT AND BUILD THE RBF SURROGATE
%*****
elseif strcmp(upper(Param.sur_type),'RBF');
%*****
% BUILD THE FUNCTION AND TIME BASED RBF SURROGATE
[Param.rbfmodelF] =
buildRBF(Param.surxp,Param.surf,Param.rbfkernel,Param.reg);
[Param.rbfmodelT] =
buildRBF(Param.surxp,Param.surtotal_time,Param.rbfkernel,Param.reg);
%*****
% UPDATE THE SURROGATE BUILDER COUNT AND THE RBF SURROGATE BUILD COUNT
Param.surbuilder_count = Param.surbuilder_count+1;
Param.rbf_surbuilder_count = Param.rbf_surbuilder_count + 1;
%*****
% IF A THE DACE-RBF SURROGATE IS BEING USED AND A DACE SURROGATE HAS
% BEEN BUILT MAINTAIN THE OPTIMAL THETA VALUE FOR THE NEXT DACE
% SURROGATE BUILD AND THE CONDITION NUMBER ASSOCIATED WITH DACE
% CORRELATION MODEL
if Param.dace_surbuilder_count >= 1
opt_theta = Param.dmodelF.theta;
condRF = condest(Param.dmodelF.R);
condRT = condest(Param.dmodelT.R);
min_cond = min(condRF,condRT);
%*****
% DISPLAY SURROGATE INFORMATION
display(['SURROGATE BUILD = ',num2str(Param.surbuilder_count),...
' (' ,upper(Param.sur_type), '
',upper(func2str(Param.reg)),')'])
display([upper(Param.sur_type), ' SURROGATES BUILT = ',...
num2str(Param.rbf_surbuilder_count), ' : THETA = ',...
mat2str(opt_theta), ' ', 'CONDITION NUMBER =
',num2str(min_cond,'%10.8e')]);
%*****
% ELSE IF THE DACE-RBF SURROGATE IS BEING USED AND A DACE SURROGATE HAS

```

```

% NOT BEEN USED MAINTAIN THETA AND CONDITION NUMBER AT THEIR INITIAL
% VALUES
else
    opt_theta = Param.theta_init;
    min_cond = Param.condnum;
    %*****
    % DISPLAY SURROGATE INFORMATION
    display(['SURROGATE BUILD = ',num2str(Param.surbuilder_count),...
            ' (',upper(Param.sur_type), '
            ',upper(func2str(Param.reg)),')'])
    display([upper(Param.sur_type), ' SURROGATES BUILT = ',...
            num2str(Param.rbf_surbuilder_count), ' : THETA = ',...
            mat2str(opt_theta), ' ', 'CONDITION NUMBER =
            ',num2str(min_cond,'%10.8e')]);
end
end
%*****
%*****
% SETS THE Param DATA THAT CHANGED IN THIS FUNCTION AND WILL BE USED
% THROUGHOUT THE NOMADm SOFTWARE
%*****
setappdata(0,'PARAM',Param);
%*****
%*****
return

%*****
function SUR_Fx=searchsurrogate_eval(x)
%*****
%*****
% SEARCHSURROGATE_EVAL IS THE SURROGATE OBJECTIVE FUNCTION TO BE OPTIMIZED
% DURING THE SURROGATE OPTIMIZATION PROCESS

% INPUT:
%   x--SET OF PARAMETERS TO BE EVALUATED BY THE SURROGATE OBJECTIVE
% OUTPUT:
%   SUR_Fx--SURROGATE OBJECTIVE VALUE RETURNED DURING SURROGATE OPT
%*****
%*****
% CALLS THE Param DATA THAT IS USED THROUGHOUT THE NOMADm SOFTWARE
%*****
Param = getappdata(0,'PARAM');
%*****
%*****
% DETERMINES THE TYPE OF SURROGATE BEING USED AND EVALUATES THE SET OF
% PARAMETERS IN THE SURROGATE APPROXIMATION MODEL
%*****
switch upper(Param.sur_type)
    case 'DACE'
        [SUR_Fx] = predictor(x,Param.dmodelF);
    case 'RBF'
        [SUR_Fx] = evalRBF(x,Param.rbfmodelF);
end
%*****

```

```

%*****
% SETS THE Param DATA THAT CHANGED IN THIS FUNCTION AND WILL BE USED
% THROUGHOUT THE NOMADm SOFTWARE
%*****
setappdata(0,'PARAM',Param);
%*****
%*****
return

%*****
function [A,l,u] = searchsurrogate_Omega(n)
%*****
%*****
% SEARCHSURROGATE_OMEGA SETS UP THE DOMAIN OF THE SURROGATE OPTIMIZATION
% PROBLEM

% INPUT:
%   n--DIMENSIONALITY OF THE PROBLEM
% OUTPUT:
%   A--IDENTITY MATRIX FOR SETTING UP THE BOUND CONSTRAINTS
%   l--LOWER BOUND OF THE CONTINUOUS PARAMETERS
%   u--UPPER BOUND OF THE CONTINUOUS PARAMETERS
%*****
%*****
% CALLS THE Param DATA THAT IS USED THROUGHOUT THE NOMADm SOFTWARE
%*****
Param = getappdata(0,'PARAM');
%*****
%*****
% SET UP THE CONSTRAINTS AND THE BOUNDS
%*****
A = eye(n);
l = Param.surLB;
u = Param.surUB;
%*****
%*****
return

```



```

%*****
% CODE FOR IMPLEMENTING SINGULAR VALUE DECOMPOSITION TO REDUCE ILL-CONDITIONING
% IN THE DACEFIT FUNCTION USED FOR OPTIMIZING THETA
%*****
function [obj,fit] = objfunc(theta,par)

% Initialize
obj = inf;
fit = struct('sigma2',NaN,'beta',NaN,'gamma',NaN,'C',NaN,'Ft',NaN,'G',NaN, ...
            'R',NaN,'E',NaN);
m = size(par.F,1);

% Set up R
r = feval(par.corr, theta,par.D);
idx = find(r > 0);
o = (1:m)';
mu = (10+m)*eps;
R = sparse([par.ij(idx,1);o], [par.ij(idx,2);o], [r(idx);ones(m,1)+mu]);

% Cholesky factorization
fullR = full(R+R'-diag(diag(R)));
[C,rd] = chol(R);

% Do SVD if R is not postive definite or it is ill-conditioned
C = C';
if rd || condest(fullR) >= 1/eps
    [U,S,V] = svd(fullR);
    s = diag(S);
    e = zeros(length(s),1);
    ind = s/max(abs(s)) >= eps;
    e(ind) = 1./s(ind);
    E = V*diag(e)*U';
    Ft = E*par.F;
    Yt = E*par.y;
    detR = prod(s.^(2/m));
else

    % Get least squares solution
    Ft = C\par.F;
    Yt = C\par.y;
    detR = prod(full(diag(C)).^(2/m));
    E = NaN;
end
[Q,G] = qr(Ft,0);

% Check ill-conditioning of G (or Ft) and F
if rcond(G) < 1e-10
    if cond(par.F) > 1e+15
        T = sprintf('F is too ill-conditioned\nPoor combination of regression model
                    and design sites');
        %error('dace:IllConditioned',T);
    else
        return
    end
end
% Do SVD if G is ill-conditioned
if condest(G) >= 1/eps

```

**DACEFIT [10]
 ALTERED TO
 USE SVD**

```

[U,S,V] = svd(G);
s       = diag(S);
e       = zeros(length(s),1);
ind     = s/max(abs(s)) >= eps;
e(ind)  = 1./s(ind);
Ginv    = V*diag(e)*U';
beta    = Ginv*(Q'*Yt);
else
    %Compute beta via LU factorization
    beta = G\ (Q'*Yt);
end

rho     = Yt - Ft*beta;
sigma2  = sum(rho.^2)/m;
obj     = sum(sigma2)*detR;
if nargout > 1
    fit = struct('sigma2',sigma2,'beta',beta,'gamma',rho'/C,'C',C,'Ft',Ft,...
                'G',G,'R',fullR,'E',E);
end
return

```

**DACEFIT [10]
 ALTERED TO
 USE SVD**

Vita

Captain David M. Bethea graduated from Hampton High School in Hampton, Arkansas. He entered undergraduate studies at Southern Arkansas University in Magnolia, Arkansas where he graduated Summa Cum Laude with a Bachelor of Science degree in Mathematics in December 2001. He was commissioned through Officer's Training School at Maxwell AFB, Alabama.

His first assignment was as a Precision-Guided Munitions Weapons and Tactics Analyst with the 86th Fighter Weapons Squadron, 53d Weapons Evaluation Group, 53d Wing, Eglin AFB, Florida. As a hard-target and PGM Weapons and Tactics expert, he developed and taught hard-target weaponeering academics to aircrew at the United States Air Force Weapons School. While stationed at Eglin, he deployed to bases in Southwest Asia to provide hard-target expertise and analytical support to the Combined Air Operations Center that resulted in recommendations to the Combined Force Air Component Commander for enhancing precision guided munitions reliability in Southwest Asia. In August 2006, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the College of Aerospace Doctrine, Research and Education, Maxwell AFB, Alabama.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 24-03-2008		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) Mar 2007 — Mar 2008	
4. TITLE AND SUBTITLE IMPROVING MIXED VARIABLE OPTIMIZATION OF COMPUTATIONAL AND MODEL PARAMETERS USING MULTIPLE SURROGATE FUNCTIONS				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
				5d. PROJECT NUMBER ENC-08-110	
6. AUTHOR(S) Bethea, David, M., Captain, USAF				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GOR/ENC/08-01	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				10. SPONSOR/MONITOR'S ACRONYM(S)	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Los Alamos National Laboratory Attn: Dr. Thomas Asaki Bikini Atoll Rd, SM 30 Los Alamos, NM 87545 (505)-667-5787 email:asaki@lanl.gov					
11. SPONSOR/MONITOR'S REPORT NUMBER(S)					
12. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT This research focuses on reducing computational time in parameter optimization by using multiple surrogates and subprocess CPU times without compromising the quality of the results. This is motivated by applications that have objective functions with expensive computational times at high fidelity solutions. Applying, matching, and tuning optimization techniques at an algorithm level can reduce the time spent on unprofitable computations for parameter optimization. The objective is to recover known parameters of a flow property reference image by comparing to a template image that comes from a computational fluid dynamics simulation, followed by a numerical image registration and comparison process. Mixed variable pattern search and mesh adaptive direct search methods were applied using surrogate functions in the search step to produce solutions within a tolerance level of experimental observations. The surrogate functions are based on previous function values and computational times of those values. The use of multiple surrogates at each search step provides parameter selections that lead to improved solutions of an objective function evaluation with less computational time. Previously computed values for the objective function and computation time were used to compute a time cut-off parameter that allows termination during an objective function evaluation if the computational time exceeded a threshold or a divergent template image was created. This approach was tested using DACE and radial basis function surrogates within the NOMADm MATLAB® software. The numerical results are presented.					
15. SUBJECT TERMS Fluid Dynamics, Image Registration, Mixed Variable Optimization, Mesh Adaptive Direct Search, Surrogate Functions, Design and Analysis of Computer Experiments, Radial Basis Functions					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Lt. Col. Mark A. Abramson, PhD, (ENC)
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			UU