SANDIA REPORT

SAND2005-7498
Unlimited Release
Printed November 2005

Developing a Computationally Efficient Dynamic Multilevel Hybrid Optimization Scheme using Multifidelity Model Interactions

Joseph P. Castro, Genetha A. Gray, Anthony A. Giunta, and Patricia D. Hough

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/ordering.htm

SAND2005-7498
Unlimited Release
Printed November 2005

Developing a Computationally Efficient Dynamic Multilevel Hybrid Optimization Scheme using Multifidelity Model Interactions

Joseph P. Castro
Electrical and Microsystem Modeling, Dept. 1437
Sandia National Laboratories
P.O. Box 5800, Albuquerque, NM 87185-0316
jpcastr@sandia.gov

Genetha A. Gray
Computational Sciences & Math, Dept. 8962
Sandia National Laboratories
P.O. Box 969, MS9159 Livermore, CA 94551-0969
gagray@sandia.gov

Anthony A. Giunta
Validation and Uncertainty Quantification Processes, Dept. 1533
Sandia National Laboratories
P.O. Box 5800, Albuquerque, NM 87185-0828
aagiunt@sandia.gov

Patricia D. Hough
Computational Sciences & Math, Dept. 8962
Sandia National Laboratories
P.O. Box 969, MS9159 Livermore, CA 94551-0969
pdhough@sandia.gov

Abstract

Many engineering application problems use optimization algorithms in conjunction with numerical simulators to search for solutions. The formulation of relevant objective functions and constraints dictate possible optimization algorithms. Often, a gradient based approach is not possible since objective functions and constraints can be nonlinear, nonconvex, non-differentiable, or even discontinuous and the simulations involved can be computationally expensive. Moreover, computational efficiency and accuracy are desirable and also influence the choice of solution method.

With the advent and increasing availability of massively parallel computers, computational speed has increased tremendously. Unfortunately, the numerical and model complexities of many problems still demand significant computational resources. Moreover, in optimization, these expenses can be a limiting factor since obtaining solutions often requires the completion of numerous computationally intensive simulations. Therefore, we propose a multifidelity optimization algorithm (MFO) designed to improve the computational efficiency of an optimization method for a wide range of applications.

In developing the MFO algorithm, we take advantage of the interactions between multifidelity models to develop a dynamic and computational time saving optimization algorithm. First, a direct search method is applied to the high fidelity model over a reduced design space. In conjunction with this search, a specialized oracle is employed to map the design space of this high fidelity model to that of a computationally cheaper low fidelity model using space mapping techniques. Then, in the low fidelity space, an optimum is obtained using gradient or non-gradient based optimization, and it is mapped back to the high fidelity space.

In this paper, we describe the theory and implementation details of our MFO algorithm. We also demonstrate our MFO method on some example problems and on two applications: earth penetrators and groundwater remediation.

Acknowledgments

Thanks to Paul Demmie and Donald Longcope for valuable discussions and help with earth penetrator models used within this document. Thanks to Katherine Fowler for providing the groundwater remediation problem and promoting our optimization method to the groundwater community. Thanks to Mike Eldred for valuable discussions and suggestions about space mapping.

Contents

Appendix

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

With the advent of massively parallel computers, computational speeds have increased tremendously. Unfortunately, the comparable scaling of numerical and model complexities in combination with overburdened machines can still result in long compute times. Moreover, many of the optimization problems that arise in engineering require a large number of computationally expensive physics-based software simulations. In some cases, the computational expense of a full-physics simulation may make the direct coupling of the simulation code to numerical optimization software infeasible.

At Sandia National Laboratories, parallel computing is an essential and well integrated function of many of the physics application codes. Moreover, the Sandia developed DAKOTA optimization software toolkit was designed to exploit massively parallel computers and take advantage of multilevel parallelism to alleviate computational times [10]. Since parallelism alone may not sufficiently reduce compute times, intelligent solution techniques and algorithms are critical for overall optimization efficiency. Thus, DAKOTA includes optimization strategies such as surrogate-based optimization (SBO). The SBO method is a two-level multifidelity optimization approach [14]. A surrogate or low fidelity model is optimized using periodic corrections from the high fidelity model. The goal of SBO is to improve the overall computational speed by using the simpler (and less computationally expensive) surrogate model. Although the SBO approach has been successful, its design precludes its application to certain problems. For example, the SBO method requires that the design variables are equivalent (one-to-one mapping) between high and low fidelity models. Therefore, the goal of our project was to design a dynamic multifidelity-multilevel hybrid optimization thereby expanding the capabilities of the DAKOTA toolkit.

To reach our ultimate goal of developing and implementing the software and algorithms for a multifidelity-multilevel hybrid optimization scheme, our original plan was to expand upon the current SBO strategy in DAKOTA. As with SBO, our idea was to use a low fidelity model to drive the optimization of the high fidelity model. For example, given two low fidelity models that capture particular physical trends (*e.g.* one displacement and the other acceleration) of a high fidelity model, the appropriate low fidelity model could be

used to drive the optimization of the high fidelity model at a given iteration. Unfortunately, two issues arise in this approach:

1. At each iteration, DAKOTA optimizes over all the design variables at once. It is not currently possible to drive the optimization of design variables separately within a single iteration. Significant changes to DAKOTA would be required to allow for this feature.

2. The number of design variables in the high fidelity model must be equal to the number of design variables in any/all low fidelity models.

By definition, low fidelity models are less descriptive than their high fidelity counterparts. Thus, high fidelity models may include design variables that are not a part of their corresponding low fidelity models (*e.g.* another physical dimension). Therefore, the optimization of the high fidelity model cannot be restricted to identifying only those variables that are also present in the low fidelity model. Given these difficulties, we opted to formulate an approach to multifidelity optimization (MFO) independent of the existing SBO method.

The theoretical details of our MFO scheme are discussed in Chapter 2. Two of the most important features of our approach are

- The algorithm is provably convergent (see Section 2.3).

- The number of design variables in the models of varying fidelity need not be equivalent in number or type. (see Section 2.2).

Designing a scheme that is provably convergent is important because the existence of convergence lends credibility and separates our approach from heuristic methods with no theoretical basis. Supporting models with different numbers of design variables expands the applicability of our method, particularly within Sandia. At Sandia, the design of physics-based codes often begins with a research level code that includes robust and complex physics but simple mathematical models. The limitation of this type of code is often a simplistic geometry and/or the lumping of physical processes (*e.g.* averaging over spatial dimensions). Therefore, for critical applications, research codes are improved by adding more complex mathematical models, solver algorithms, software infrastructures, etc. The result is diverse set of software models for the same physical process with different complexities and compute times. Our MFO approach lends itself particularly well to this hierarchical development process. Often, as codes are developed, design spaces become better defined and design variables are added. Appropriate MFO schemes for applications that involve such codes will not place any limitations on the number of design variables in each model.

In this report, we describe the development, implementation, and testing of an algorithmic approach to MFO. Our method takes advantage of interactions between multifidelity

models to develop a dynamic and computationally time-saving algorithm. In Chapter 2, we review the theoretical basis of our method and describe how existing convergence results can be applied to our MFO algorithm. Chapter 3 includes all the details related to implementation. A set of test problems and the results we obtained are given in Chapter 4. Testing of our MFO algorithm is expanded to engineering applications in Chapter 5. We explain two problems and exhibit the success of our MFO algorithm in solving them. Finally, in Chapter 6, we give some final thoughts. Note that the Appendix includes an example of the scripting process used in the implementation of one of our examples.

Chapter 2

Developing a Multifidelity-Multilevel Hybrid Optimization Scheme

The development of our MFO scheme is based on the theory of a direct search algorithm scheme and space mapping techniques.

2.1 Asynchronous Parallel Pattern Search (APPS)

The MFO approach described in this paper incorporates a derivative-free optimization method called Asynchronous Parallel Pattern Search (APPS) [18, 19]. The APPS algorithm is part of a class of direct search methods which were primarily developed to address problems in which the derivative of the objective function is unavailable and approximations are unreliable [37, 25]. Pattern searches use a predetermined pattern of points to sample a given function domain. It has been shown that if certain requirements on the form of the points in this pattern are followed and if the objective function is suitably smooth, convergence to a stationary point is guaranteed [9, 24, 35].

The majority of the computational cost of pattern search methods is the function evaluations, so parallel pattern search (PPS) techniques have been developed to reduce the overall computation time. Specifically, PPS exploits the fact that once the points in the search pattern have been defined, the function values at these points can be computed simultaneously [8, 34]. The APPS algorithm is a modification of PPS that eliminates the synchronization requirements. It retains the positive features of PPS, but eliminates processor latency and requires less total time than PPS to return results [18]. Implementations of APPS have minimal requirements on the number of processors and do not assume that the amount of time required for an objective function evaluation is constant or that the processors are homogeneous.

In this work, we consider the specific APPS algorithm as described in [19]. It is a

variant on generating set search as described in [20] and is provably convergent under mild conditions [21, 22, 19]. Omitting the implementation details, the basic APPS algorithm can be simply outlined as follows:

1. Generate a set of trial points **T** to be evaluated.
2. Send the set **T** to the *conveyor* for evaluation, and collect a set of evaluated points, **E**, from the conveyor. (The conveyor is a mechanism for shuttling trial points through the process of being evaluated.)
3. Process the set **E** and see if it contains a new *best point*. If **E** contains such a point, then the iteration is successful; otherwise, it is unsuccessful.
4. If the iteration is successful, replace the current best point with the new best point (from **E**). Optionally, regenerate the set of search directions and delete any pending trial points in the conveyor.
5. If the iteration is unsuccessful, reduce certain step lengths as appropriate. In addition, check for convergence based on the step lengths.

A detailed procedural version of APPS is given in [15], and a complete mathematical description and analysis is available in [19]. APPSPACK version 4.0 is a software implementation of this algorithm. In Section 3.2, we discuss some of its basic implementation details and describe the customizations made specifically for our MFO procedure.

2.2 Space Mapping

Space mapping [4, 5] is a numerical technique that allows linking design spaces of models with similar functionality (*i.e.* modeling the same physical process), but varying fidelities (*i.e.* differing dimensions, differing physics assumptions). In this report, we refer to these models of differing fidelity as the high fidelity model and the low fidelity model. The high fidelity model is considered to be the "best" or more exact model but is computationally more expensive. In contrast, the low fidelity model is less exact but requires fewer computational resources. The relationship between these two models can be defined by a mapping P such that the high fidelity model design space, $\vec{x_H}$, and the low fidelity design space, $\vec{x_L}$, are related via the equation

$$\vec{x_L} = P\left(\vec{x_H}\right) \tag{2.1}$$

such that

$$f_L\left(P\left(\vec{x_H}\right)\right) \approx f_H\left(\vec{x_H}\right) \tag{2.2}$$

where f_H and f_L are the high and low fidelity model responses respectively. Equation (2.2) can be restated as a minimization problem of the form

$$min \left\{ \sum_{i=1}^{N} \| f_L(P(x_i)) - f_H(x_i) \|^2 \right\} \tag{2.3}$$

where N is the number of high fidelity design points x_i. (This is done with a least square technique, see Section 3.6.1 for details.) Finding an appropriate formulation for P is highly dependent on the problem type. One mapping that we have consistently used is of the form

$$P(x_H) = \alpha x_H^{\beta} + \gamma \tag{2.4}$$

where α, β, and γ are determined by solving (2.3). This formulation is in no way a complete and generic mapping, nor is it our only option, but it is an appropriate starting point for our research.

It is also important to note that $\vec{x_H}$ is not restricted to being equivalent to $\vec{x_L}$. This is an important feature of space mapping since it is often the case that a low fidelity model has fewer design parameters to characterize than a corresponding higher fidelity model. Many MFO schemes require a one-to-one correspondence between the design space of models. We incorporate space mapping into our MFO approach to eliminate this restriction. Section 4.2 includes examples of models with an unequal numbers of design variables and their corresponding mappings.

2.3 The APPS/Space Mapping Scheme

The APPS/Space Mapping approach to MFO can be described in terms of two loops– an outer loop and an inner loop. The main purpose of the outer loop is the application of the APPS algorithm to the high fidelity model. However, it also has the added task of maintaining a set of trial points and their corresponding response values. These are used by the inner loop to map the high fidelity space to the low fidelity space. The inner loop then uses this space mapping to optimize the low fidelity model. The complete MFO procedure is:

1. Start the outer loop.
    - Optimize the high fidelity model f_H using the APPS algorithm.
    - While optimizing, collect a set of N pairs $(x_i, f_H(x_i))$
2. Start the inner loop

- Using the N high fidelity response pairs collected by the outer loop, obtain the space mapping parameters. In other words, find α, β, and γ such that

$$\sum_{i=1}^{N} \left\| f_L(\alpha(x_i)^{\beta} + \gamma) - f_H(x_i) \right\|^2. \tag{2.5}$$

  is minimized (see Equation 2.3). See Section 3.6.1 for a discussion of the least squares optimization algorithm applied to solve (2.5).

- Optimize the low fidelity model within the space mapped high fidelity space by minimizing $f_L(\alpha x^{\beta} + \gamma)$ with respect to x; obtain x^*.

3. Return x^* to the APPS algorithm and determine if it is a new best point.

This algorithm is illustrated in Figure 2.1, and its implementation is discussed in Chapter 3 and in the Appendix.

One of the main theoretical considerations in developing an MFO method is convergence. As discussed in Section 2.1, the APPS algorithm is provably convergent under mild conditions [21, 22, 19]. This result can be leveraged by the APPS/Space Mapping scheme if we view the inner loop as an *oracle*. In optimization methods, oracles are used to predict points at which a decrease in the objective function might be observed. Analytically, an oracle is free to choose points by any finite process. (See [20] and references therein.)

In the case of the APPS/Space Mapping scheme, the inner loop acts as an oracle in that it chooses additional candidate points for minimizing the high fidelity model. Note these points are given in addition to those generated by the pattern search of outer loop. Therefore, the convergence of the APPS algorithm is not adversely affected by the addition of the inner loop. All other changes required in the implementation of the APPS algorithm to incorporate the inner loop are merely cosmetic and do not affect the underlying algorithm. Thus, no additional work is required to prove convergence of the APPS/Space Mapping approach to MFO. However, future work may include investigating any improvement to the convergence of the APPS algorithm that may be attained from the addition of the inner loop described here.

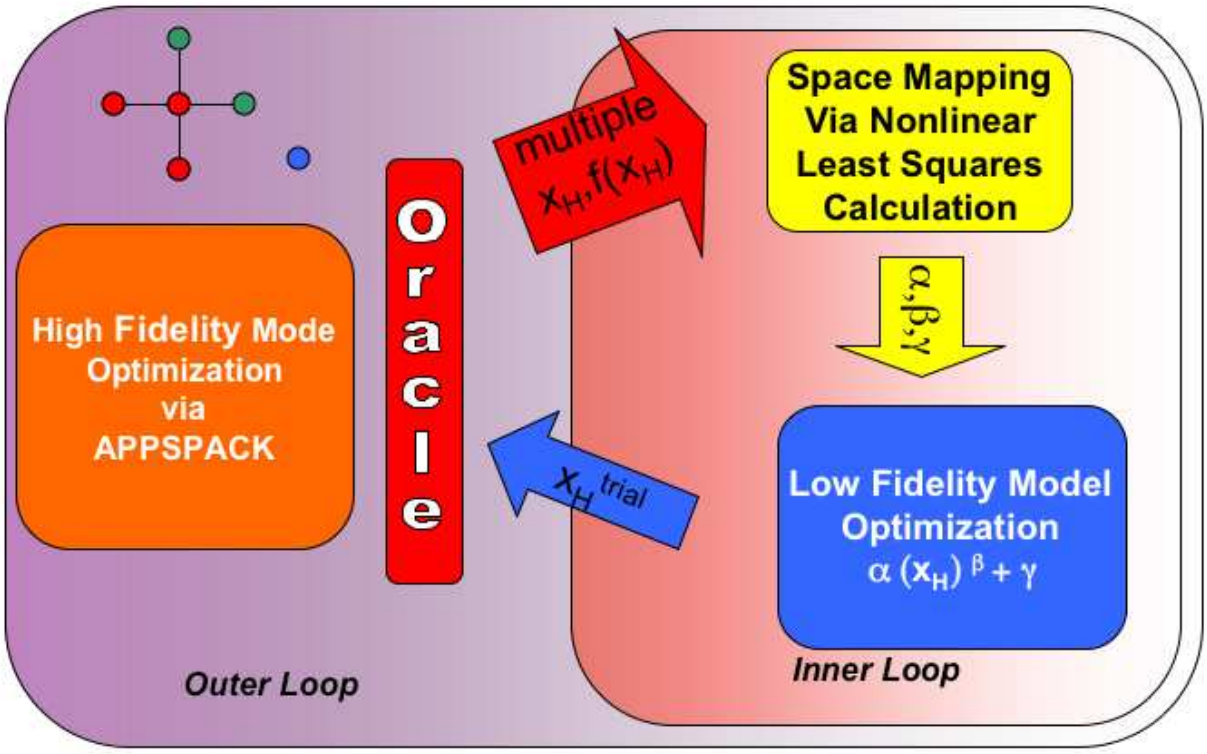

Figure 2.1. The APPS/Space Mapping Scheme: First, APPSPACK is applied to the high fidelity model over a reduced design space. In conjunction with the search, a specialized oracle is employed to map the design space of this high fidelity model, a set of $(x_H, f(x_H))$ pairs, to that of a computationally cheaper low fidelity model using space mapping techniques. Using the resulting space mapping parameters, α, β, and γ, an optimum is obtained in the low fidelity design space. The result, x_H^{trial}, is returned to APPSPACK as a possible new best point.

Chapter 3

Software Implementation of the APPS/Space Mapping Scheme

The APPS/Space Mapping scheme was implemented using an existing software implementation of the APPS algorithm and a series of customized scripts.

3.1 The DAKOTA Toolkit

Much of the existing software that we will discuss in this Chapter is included DAKOTA (Design and Analysis Toolkit for Optimization and Terascale Applications) [10], an object-oriented suite of analysis and optimization methods under development at Sandia National Laboratories. DAKOTA is also used extensively in the customized MFO inner loop script. Therefore, we provide a short overview of the DAKOTA toolbox here.

DAKOTA provides a flexible framework for conducting parameter estimation, sensitivity analysis, uncertainty quantification, design of experiments sampling, and optimization. Included in the optimization tools are methods for solving continuous, discrete, and mixed continuous-discrete problems. The analysis and optimization methods in DAKOTA are designed to exploit massively parallel computers (typically having $O[10^3 - 10^4]$ processors) which were developed under the Department of Energy's Accelerated Strategic Computing Initiative (ASC). While intended for MP computers, DAKOTA also may be used on a single workstation or on a cluster of workstations. To date, DAKOTA has been ported to most common UNIX-based workstations including Sun, SGI, DEC, IBM, and LINUX-based PCs.

3.2 APPSPACK-4.0: Software Implementation of APPS

One optimization routine included in DAKOTA is APPSPACK, a software implementation of the APPS algorithm discussed in Section 2.1. It is targeted at simulation-based optimization. These problems are characterized by a relatively small number of variables (*i.e.* $n < 100$), and an objective function whose evaluation requires the results of a complex simulation. Because, APPS is a direct search method, gradient information is not required. Therefore, APPSPACK is applicable to a variety of problems. Moreover, the procedure for evaluating the objective function can be executed via a separate program or script. This simplifies its execution and makes it amenable to customization.

APPSPACK version 4.0 is a software implementation of the specific APPS algorithm presented in [19]. It is written in C++ and uses MPI [16, 17] for parallelism. The details of the implementation are described in detail in [15]. There are both serial and parallel versions of APPSPACK-4.0, but to achieve the goals in developing an MFO algorithm, we are solely interested in the parallel version. Of particular interest to us is the management of the function evaluation process. The procedure is actually quite general and merely one way of handling the process of parallelizing multiple independent function evaluations and efficiently balancing computational load. This makes the code applicable in a wide variety of contexts and easily customizable. In this section, we review some of the basic features and how we customized them to fit our MFO procedure.

3.2.1 Point Objects

In the APPSPACK software, points are stored as objects that contain information about how the point was generated and its function value (if it has been evaluated). In general, new trial points are generated using a predetermined pattern and the current best point. For the MFO algorithm, some additional trial points are generated using the oracle or inner loop as described in Section 2.3. The point objects in APPSPACK also store *state* information that indicates whether or not the point has been evaluated and, if it has, whether or not it satisfies certain decrease conditions. We customized this state field for the MFO algorithm so that it also includes information about how this point was generated (*i.e.* by the oracle or otherwise) and whether or not it should be used as a starting point for a future inner loop calculation.

3.2.2 Evaluation Conveyor

The basic purpose of the evaluation conveyor is to exchange a set of unevaluated points **T** is for a set of evaluated points **E**. The set **T** may be empty, but the set **E** must be non-empty because returning an empty set of evaluated points means that the current iteration cannot proceed. More specifically, the evaluation conveyor facilitates the movement of

points through the following three queues:

- **W**, the "Wait" queue where trial points wait to be evaluated.
- **P**, the "Pending" queue for points with on-going evaluations. Its size is restricted by the resources available for function evaluations.
- **R**, the "Return" queue where evaluated points are collected. Its size can be controlled by the user.

Note that it may take more than one APPSPACK iteration (e.g. the return of more than one non-empty set of evaluated points **E**) for a point to move through the conveyor. Therefore, at each iteration, the input set **T** will not necessarily contain the same points as the output set **E**. Furthermore, depending on resources, it may be desirable to *prune* (*i.e.* remove) some or all of the points that are in the **W** queue.

Points stored in **W** remain there until either there is space in **P** or they are pruned. By default, the **W** queue is pruned whenever a new best point is found, and pruning means emptying all the points currently still in **W**. For the MFO algorithm, we modify the pruning process by adjusting one of the existing APPSPACK solver parameters, inherent to the algorithm. Although the points remaining in the queue are not good candidates for progressing the optimization in the MFO outer loop, the evaluation results are still needed in the inner loop (or oracle) process.

It should also be noted that the normal process for adding points to **W** is to put the newest point at the end of the list. We customize this procedure for the MFO algorithm by placing points that will initiate an inner loop calculation at the beginning of the queue. Since such points are more likely to be an improvement on the current best point, we want to evaluate them as soon as resources are available.

Before a point is pushed from **W** to **P**, certain criteria must be met. In the case of the MFO algorithm, these criteria are based on whether or not a point requires an inner loop calculation. If there is to be no inner loop calculation, the criteria is a cache check. Basically, if the function value has already been calculated for that point, it is obtained from the cache and the corresponding point is moved directly to **R**. Otherwise, the point moves to **P** and is evaluated. If the point does require an inner loop calculation, an appropriate sets of N high response pairs must be available. If such a set exists, then the point moves to **P** and is evaluated. If the set has not yet been assembled, the point remains in **W** until the set of pairs is completed. It is important to note that once a point is removed **W** and placed in **P** or **R**, it cannot be pruned.

Points which have been submitted to the executor for evaluation are stored in **P**. The size of **P** depends on the available resources. Because function evaluations are often computationally expensive, points may remain in **P** for several iterations. Once the executor returns the results of the function evaluation, the point is moved to **R**.

The conveyor process continues until **R** reaches a user specified size. The default size is one, but larger values can be used. In the extreme, a synchronous pattern search [23] can be defined by requiring every trial point to be evaluated and collected. However, since the MFO algorithm was designed to take advantage of the asynchronous characteristics of APPSPACK, we use the default size for **R**.

3.2.3 Function Executor and Evaluator

Once a point is pushed to the pending queue, **P**, it must be assigned to a worker and evaluated. It is the job of the executor to coordinate the assignment of points to workers for function evaluation.

The evaluator is its own entity and is not actually part of the executor. Each worker owns its own evaluator object and receives messages from the executor on the master processor with the information it needs to pass on to the evaluator. Therefore, in the MFO algorithm, we have customized the executor on the master processor so that it not only sends the point to be evaluated but also a message indicating whether or not an inner loop calculation should be done.

The actual objective function evaluation of the trial points is handled by the evaluator. For the MFO algorithm, it works as follows: A function input file containing the point to be evaluated and a simple message indicating whether or not an inner loop calculation should be done is created. Then, an external system call is made to the user-provided executable that calculates the function value. This executable is described in more detail within Section 3.3. After the function value has been computed, the evaluator reads the result from the function output file. In addition, the evaluator reads a message from the function output file which describes how this value was obtained (*i.e.* with or without an inner loop calculation). Finally, both the function input and output files are deleted.

By default, APPSPACK function evaluations are run as separate executables, and communication with the evaluation executable is done via file input and output. In other words, each worker makes an external system call. This ensures applicability of APPSPACK to simulation-based optimization and eliminates any requirements that simulators be encapsulated into a subroutine. In the case of MFO, we leverage this design and provide a script that first runs an inner loop if called for and then calculates the high fidelity function value using either the results from inner loop or the point contained in the APPSPACK function input file. This is described in detail in Section 3.3.

3.2.4 Cache

Because the APPS algorithm is based on searching a pattern of points that lie on a regular grid, the same point may be revisited several times. To avoid evaluating the objective

function at any point more than once, APPSPACK employs a function value cache. Its functions include inserting new points into the cache, performing lookups, and returning previously calculated function values. Optionally, the cache manager can also create an output file with the contents of the cache or read an input file generated by a previous run.

In the case of the MFO algorithm, the procedures for inserting new points into the cache, performing lookups and retrieving previously cached values remain the same. They are described in detail in [15]. Note that each time a function evaluation is completed and a trial point is placed in the return queue **R**, the conveyor stores this point and its corresponding function value in the cache. And, as previously discussed, before sending any point that does not require an inner loop calculation to the pending queue **P**, the conveyor first checks the cache to see if a value has already been calculated. If it has, the cached function value is used instead of repeating the function evaluation.

The customizations made to the cache for the MFO algorithm relate to the optional cache output file. Entries in the default cache output file only include the point and its function value. Our cache output files includes those things and some of the additional information stored in the APPSPACK point structure. This extra information allows us to appropriately gather the sets of N high fidelity response pairs needed by the inner loop. Moreover, allowing the function evaluation script to access the cache using the output file greatly simplifies the implementation and execution of the MFO algorithm.

3.3 Scripts for the APPS/Space Mapping Function Evaluation

As previously discussed, APPSPACK performs function evaluations through system calls to an external executable, and no restrictions are placed on the language of this executable. For the APPS/Space Mapping approach to MFO, a customized csh/Perl script was created to manage the function evaluation process. Basically, it carries out the following steps:

- Read the trial point x and corresponding message from the APPSPACK function input file.
- If message = "yes" and inner loop conditions are met,
  - Run inner loop script (as described in section 3.6).
  - Replace x with the point returned from the inner loop.
- Run high fidelity script.
  1. Copy high fidelity template directory into temporary work directory.
  2. Create high fidelity input file with template input file and trial point x.
  3. Evaluate high fidelity function; may include calls to simulators, etc.

4. Write the resulting function value and corresponding message to the APPSPACK function output file.

For a specific example of such a script, see Appendix A.1.

The high fidelity script oversees the execution of the high fidelity function evaluation. It was written to be as general as possible, so user modification is necessary to accommodate specific applications. As seen in the example high fidelity script in the Appendix, our script is explicitly labeled to indicate which sections require user modification. Specifically, the user must add the appropriate command line call for the external high fidelity function executable of interest. None of the other tasks performed by the high fidelity script require any user modification

Completing a high fidelity function evaluation usually requires access to additional files and directory structures. For example, obtaining a high fidelity response value may entail running a simulator or mesh generator. Therefore, a high fidelity template directory is maintained to store the pertinent information. The high fidelity script copies this directory into a unique temporary work directory to prevent potential file overwriting. In addition, the template directory contains a sample input file with aprepro [32] place holders for the components of x. During the function execution, aprepro is used to insert the current values into these place holders and generate an appropriate input file for the current executable run. Examples of both a high fidelity script and corresponding high fidelity template directory structure are given in Appendix A.1.

3.4 Script Dependencies

To reasonably implement our APPS/Space Mapping MFO scheme, we developed a series of interrelated scripts. In this section, we give a high level overview of these scripts and their dependencies. The entire MFO scheme is outlined in Figure 3.1. This figure gives a graphical representation of the algorithm and includes APPSPACK, the scripts, and the process of algorithm execution. It also illustrates how the scripts are related to one another.

The primary run script is *MFO_RUN_SCRIPT*. It is executed directly by the user to initiate the entire MFO algorithm. This script also includes some preprocessing steps that ensure that the APPSPACK input file and the *USER_INTERFACE_SCRIPT* have the appropriate dependencies (*e.g.* , the APPSPACK input file must specify USER_SCRIPT as the function executable). For each APPSPACK trial point, a temporary *USER_SCRIPT* is generated that contains all the relevant function evaluation information. Next, this USER_SCRIPT generates a temporary interface script file, *interface_script_$tag*,whose job is to obtain the most current APPSPACK information (primarily high fidelity function values from the APPSPACK cache file). The interface script then calls the primary inner loop control script, *mfo_script*. Using the criteria discussed in Section 3.6, the least squares evaluation (*least_squares_script*), the space mapped low fidelity optimization cal-

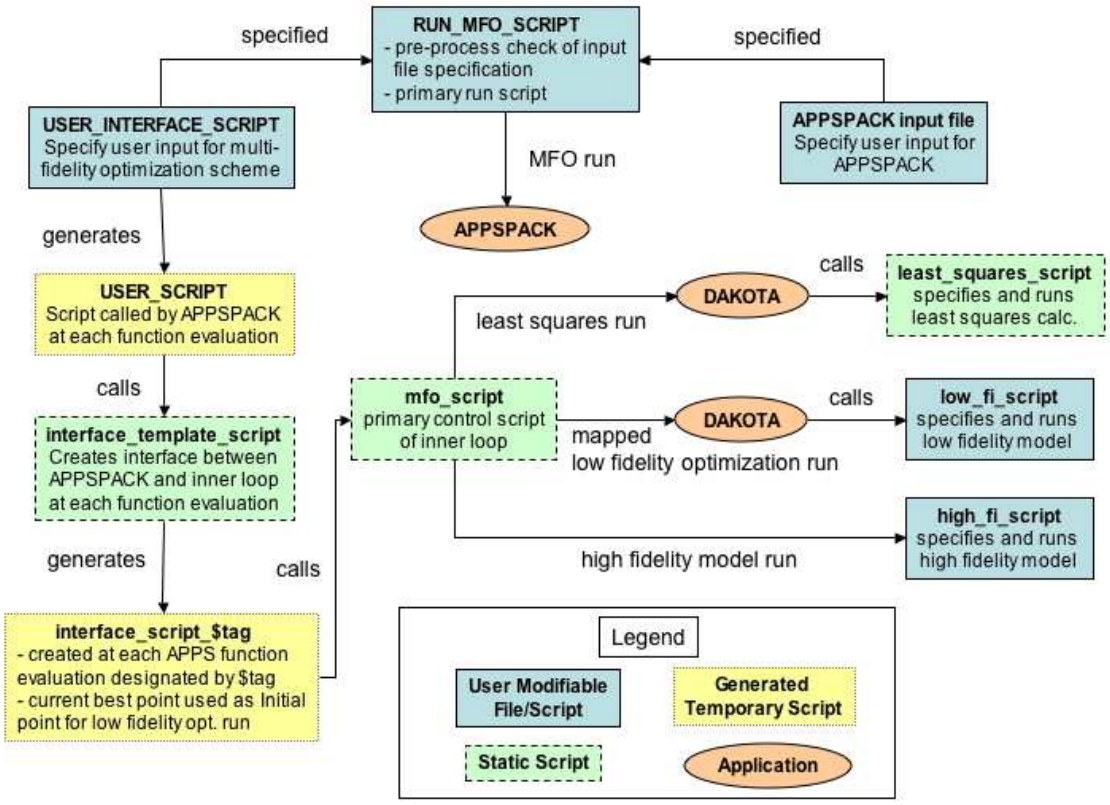

Figure 3.1. A high level view of the implementation of the MFO scheme and its script dependencies.

culation (*low_fi_script*), and/or the high fidelity response calculation (*high_fi_script*), are subsequently executed. More details of these scripts are given in the remainder of this Chapter.

3.5 Scripts for User Interfacing

Our MFO scheme was implemented in such a way as to keep user modifications to a minimum. However, it was necessary to also keep in mind the special needs of specific applications and their external executables. Thus, the run-time options available to the user include some control of the inner loop execution, but they also provide some functionality for automating some of the otherwise tedious tasks. In the script *USER_INTERFACE_SCRIPT*, the user can opt to specify the following:

- num_responses – number of high fidelity function values required to execute an inner loop calculation.
- min_calcs – minimum number of high fidelity function calculations performed between inner loop calculations.
- high_var_tag_option (<sequential|explicit>) – definition of the high fidelity variable tags in the high fidelity template input file (see Section 3.3).
  - sequential – the single tag name defined in high_var_tag will have a sequential number (starting at 0) appended to the end of the tag.
  - explicit – all tags are defined by the user (in high_var_tag).
- high_var_tag – if sequential, a single tag value; if explicit, tag numbers in array (tag_1 tag_2 $\cdots$).
- low_var_tag_option (<one_to_one|sequential|explicit>) – definition of the low fidelity variable tags in the low fidelity template input file (see Section 3.6.2).
  - one_to_one – one-to-one mapping from high to low so parameters are defined equivalently.
  - sequential – single tag name defined in low_var_tag will have a sequential number (starting at 0) appended to the end of the tag.
  - explicit - all tags are defined by the user (in low_var_tag).
- num_low_var – number of low fidelity model variables; Note that this need not be equivalent to the number of high fidelity variables and must be set except when the one_to_one option is set.
- low_var_tag – if sequential, single tag value; if explicit, tags given in arrays (tag_1 tag_2 $\cdots$); if one_to_one, not needed.
- space_map_tag_option (<global|explicit>) – space mapping tags in low fidelity model.
  - global – low/high/init values are equivalent for a given space_map_tag for all variables.
  - explicit – all tags must be defined by the user in space_map_tag and for all associated low/high/init values. Note that each design variable is assigned to each space param tag.
- num_space_map – number of space mapping parameters; if global, no need to set this parameter.
- space_map_tag – naming scheme for space mapping parameters.
  - If global, define each space_map_tag without the high_var_tag appendix. (*e.g.* (alpha, beta)).

  - If explicit, every space_map_tag with the high_var_tag appendix must be defined. (*e.g.* (alpha_x0 alpha_x1 beta_x0 beta_x1))

- space_map_low – lower bounds of space_map_tag.
  - If global, define each value in the order of space_map_tag. (*e.g.* (alpha, beta) $\Rightarrow$ (0.0 1.0))
  - If explicit, define each value in the order of space_map_tag. (*e.g.* (alpha_x0 alpha_x1 beta_x0 beta_x1) $\Rightarrow$ (0.0 0.0 1.0 1.0))
- space_map_high – upper bounds of space_map_tag. (See space_map_low for specifications.)
- space_map_init – initial value of space_map_tag. (See space_map_low for specifications.)
- opt_strategy (<grad|apps>) – optimization strategy used in the space mapped low fidelity optimization calculation.
  - grad – gradient based method (from DAKOTA).
  - apps – traditional APPSPACK.

An example user interface script is included in Appendix A.2.1.

3.6 Inner Loop Implementation

The inner loop implementation was designed to maintain the flexibility and black box approach of the the optimization toolbox DAKOTA. It will become evident in this section that DAKOTA is the workhorse of the inner loop. Therefore, it is critical that the two be well integrated. The inner loop script carries out the following steps:

- Check cache output file validity (# evaluations $\geq$ # x points) and required number of calculations between inner loop calls (set by user).
- If all inner loop conditions are met,
  - Run the MFO script, *mfo_script*.
    1. Incorporate user modifications defined in the user interface script (described in Section 3.5).
    2. Convert any user supplied data and APPSPACK data to aprepro format.
    3. Run normalization calculation (optional, see Section 3.5).
    4. Run least squares script to calculate space mapping parameters (as described in Section 3.6.1).

5. Optimize the space mapped low fidelity model via the low fidelity optimization script (as described in Section 3.6.2).
6. Write x, the point of optimum value, to an output file.

– Using x as a trial point, run the high fidelity script (described in Section 3.3).

The MFO script is the heart of the inner loop. It does not change from application to application and requires no user customizations. Instead, all run time user options are set in the user modification scripts. Appendix A.3 includes the MFO script used for this project.

3.6.1 Computing the Space Mapping Parameters: A Series of Least Squares Calculations

There are two methods for calculating the space mapping parameters– dependent and independent. The independent approach computes the space mapping parameters for a single design variable while holding all other design variables and their corresponding space mapping parameters constant. One advantage of this method is that the the number of high fidelity responses required to complete the calculation is merely the number of space mapping parameters for a given design variable. However, calculating the space mapping parameters independently requires a least squares calculation for each design variable. Moreover, holding the other design variables constant restricts the sampling of space. In contrast, the dependent method solves for space mapping parameters of all the design variables in a single least squares calculation and does a more thoroughly sampling of the space. But, this approach requires many more function response values; the number of space mapping parameters multiplied by the number of design variables is needed. Our MFO implementation offers both methods for calculating the space mapping parameters but sets the dependent method as the default because it requires far fewer high fidelity function response values. In many of the applications of interest at Sandia, the cost of computing a high fidelity function response is often a limiting factor. Thus, we are most interested in methods that reduce the overall number of high fidelity function evaluations.

As discussed in Section 3.2.4, once a high fidelity response is computed, its value and corresponding design variables are cached in a file. If normalization is required, these cached design points are used to calculate the corresponding low fidelity responses. Then, a normalization parameter is calculated using the equation

$$\eta = \sum_i \frac{f_H(x_i)}{f_L(x_i)}, \tag{3.1}$$

where x_i is a point in the high fidelity space. Note that the low fidelity response can be normalized. In this case, a product ηf_L is used in the least squares calculation instead of f_L.

To solve (3.1), we apply the NL2SOL algorithm [13], a least squares method that is included in the DAKOTA toolkit. Our least squares script, *least_squares_script*, is called internally by DAKOTA. The primary function of this script is to facilitate the calculation of the space mapping parameters using the N high response pairs collected by the outer loop. In other words, the parameters α, β, and γ are calculated such that

$$\sum_{i=1}^{N} \left\| f_L(\alpha(x_i)^\beta + \gamma) - f_H(x_i) \right\|^2.$$

This process is implemented via a script that carries out the following steps

- Read N high fidelity responses and corresponding design variables from the APPSPACK cache file.
- Copy the low fidelity template directory and low fidelity script into a temporary work directory.
- Perform DAKOTA least squares calculations.
  - Make multiple calls to least square script which returns $\eta f_L(\alpha(x_i)^\beta + \gamma) - f_H(x_i)$.
  - Return the parameters α, β, and γ that result.
- Place DAKOTA results for the space mapping parameters into an aprepro file.

3.6.2 A Script for the Space Mapped Low Fidelity Optimization

Once the space mapping parameters have been calculated, an optimization problem is solved in the mapped low fidelity space. As described in Section 3.5, the optimization scheme applied can be specified by the user. In fact, there are no restrictions on the type of optimization scheme that can be used here, and the only consideration is implementation. Although the setup of the DAKOTA optimization run is automated, some additional setup is required to create the appropriate DAKOTA optimization file. Various DAKOTA options could be moved up into the user input file so that the user can control them, but this was considered unnecessary for our current MFO implementation. The space mapping parameters are set to one by default, but can be reset to the value calculated by the least squares method via an aprepro call. In summary, the space mapped low fidelity optimization calculation is carried out as follows:

- Read the space mapping parameters from the least squares calculation.
- Copy the low fidelity template directory and low fidelity script into a temporary work directory

- Carry out the DAKOTA space mapped low fidelity optimization calculation.
  - Makes multiple calls to the low fidelity script and read the space mapping parameters (α, β, and γ) via aprepro.
  - Return the solution, x^*.
- Place x^* into an aprepro file.

An example of the low fidelity script is included in Appendix A.2.2. The space mapped low fidelity optimization is integrated into the main MFO script and is given in Appendix A.3.

Chapter 4

Test Problems

A set of test problems was developed for two reasons:

1. To show proof of concept for our MFO scheme, and to compare optimal solutions obtained from our MFO scheme and traditional APPSPACK optimal as well as the number of high fidelity runs required by each method to reach a solution.

2. To study the sensitivity of the space mapping parameters to algorithmic input including:

   - the number of high fidelity responses used for the mapping.
   - the scaling of the space mapping parameters (or the size of the offset between the low and high fidelity models).
   - different starting points.

The two types of test functions in the test set are: (i) polynomial and (ii) multi-variable Rosenbrock [31] The polynomial function was included to examine direct mappings between the low and high fidelity models as well as space mapping sensitivity. The Rosenbrock function [30] is well known and often used by the optimization community to test new algorithms. The multi-variable Rosenbrock is a variant that can be represented by N design variables. It allows us to test the mapping between models with different numbers of design variables. Note throughout this section the mapping terms have been annotated with a * to distinguish them from terms within the test functions.

4.1 Polynomial Function

The high fidelity model of the first polynomial test function is

$$f_H(x_0,x_1) = \left(\alpha_0 x_0^{\beta_0} + \gamma_0\right)^2 + \left(\alpha_1 x_1^{\beta_1} + \gamma_1\right)^2, \tag{4.1}$$

and the corresponding low fidelity model is

$$f_L(y_0,y_1) = (y_0)^2 + (y_1)^2. \tag{4.2}$$

A direct mapping between these two models can be represented as $P(x_H)$ where

$$P(x_H) = \alpha_* x_H^{\beta^*} + \gamma^*, \tag{4.3}$$

so that the the mapped low fidelity function has the form

$$f_{mapped} = f_L(P(x_0),P(x_1)) = \left(\alpha_0^* x_0^{\beta_0{}^*} + \gamma_0^*\right)^2 + \left(\alpha_1^* x_1^{\beta_1{}^*} + \gamma_1^*\right)^2. \tag{4.4}$$

Therefore, the ideal case is

$$f_{mapped} = f_H \Rightarrow \alpha_0 = \alpha_0^*; \beta_0 = \beta_0^*; \gamma_0 = \gamma_0^*; \alpha_1 = \alpha_1^*; \beta_1 = \beta_1^*; \gamma_1 = \gamma_1^*. \tag{4.5}$$

Again, note that the mapping terms have been annotated with a * to distinguish them from the polynomial terms.

A series of calculations was done in which the order of magnitude of the mapping parameters, number of high fidelity responses to calculate the mapping, and starting point were varied. This document includes the results for the following sets of conditions:

Test 1: $\gamma \sim O(1); \alpha, \beta = 1$

Test 2: $\gamma \sim O(10); \alpha, \beta = 1$

Test 3: $\gamma \sim O(100); \alpha, \beta = 1$

Test 4: $\alpha, \gamma \sim O(1); \beta = 1$

Test 5: $\alpha \sim O(10), \gamma \sim O(100); \beta = 1$

Test 6: $\alpha \sim O(100), \gamma \sim O(10000); \beta = 1$

The results of each test case are included in the next sections. For each test case, results are displayed in both a table and a graph. The columns of the tables are the same for each test case and are as follows:

 # Hi-Fi Resp.: the number of high fidelity responses used by the inner loop for the space mapping calculation. For comparison, the last row of the table, labeled "APPS", displays results obtained using traditional APPSPACK, without making any calls to the inner loop.

$\gamma_0, \gamma_1, \alpha_0, \alpha_1$**:** calculated values of the space mapping parameters in (4.4). The true value is specified as part of the the column label. Note that not all the test cases use all four parameters.

x_0, x_1**:** final value of the design variables in (4.4). The true solution is given as part of the column label.

Objective Value: minimum calculated response (*i.e.* minimum of f_H obtained by the algorithm)

 # Hi-Fi Calc.: total number of high fidelity response calculations completed while searching for the minimum of f_H.

X Speed Imp: speed improvement of the MFO algorithm from traditional APPSPACK where speed refers to the number of high fidelity response calls that are required to complete the run. The value was calculated using the equation

$$\text{X Speed Imp} = \frac{\text{\# of high fidelity response calculations for MFO}}{\text{\# of high fidelity response calculations for APPS}}$$

The graph illustrates the algorithmic progression of each row in the table. The x-axis indicates the number of successful APPSPACK iterations where a successful iteration is defined by the identification of a new best point. (Recall that an APPSPACK best point corresponds to the point that results in the smallest objective function value.) The y-axis corresponds to the value of the objective function. The legend indicates which color corresponds to which number of high fidelity responses was used by the inner loop. The starred points were obtained without the inner loop while the circle points were the result of an inner loop calculation. (Note that although the point resulting from the first iteration is a circle, this was not obtained using the inner loop. It is merely an artifact of the plotting routine.)

4.1.1 Polynomial Test 1: $\gamma \sim O(1)$; $\alpha, \beta = 1$

This test uses the high fidelity function with the exact form

$$f_H(x_0, x_1) = (x_0 + 0.5)^2 + (x_1 - 0.83)^2.$$

The results of the MFO calculation are displayed in Table 4.1 and Figure 4.1. In this case, the space mapping parameters are nearly an exact match regardless of the number of high fidelity responses used by the inner loop. The MFO scheme provides a 2-3 times speed up from traditional APPSPACK. The graph indicates that this speed up is the result of doing an inner loop calculation early in the execution of the algorithm. Moreover, the MFO algorithm finds a better minimum.

4.1.2 Polynomial Test 2: $\gamma \sim O(10)$; $\alpha, \beta = 1$

In this case, the high fidelity function has the exact form

$$f_H(x_0, x_1) = (x_0 + 5.0)^2 + (x_1 - 8.3)^2 .$$

The test results are given in Table 4.2 and illustrated in Figure 4.2. As was the case for test 1, the space mapping parameters are nearly an exact match in all the cases. In addition, they scale well with the polynomial described in test case 1. The same speed up and improvement in algorithm progression are observed.

4.1.3 Polynomial Test 3: $\gamma \sim O(100)$; $\alpha, \beta = 1$

Test case 3 is a high fidelity function of the exact form

$$f_H(x_0, x_1) = (x_0 + 50.0)^2 + (x_1 - 83.0)^2 .$$

The results obtained with the MFO algorithm and with traditional APPSPACK are shown in Table 4.3 and Figure 4.3. The space mapping parameters are again nearly an exact match in all cases and scale very well with the test case 1 and 2 polynomials. Improvements in algorithmic progression and speed are also apparent.

4.1.4 Polynomial Test 4: $\alpha, \gamma \sim O(1)$; $\beta = 1$

For polynomial test case 4, the high fidelity function takes the exact form

$$f_H(x_0, x_1) = (0.8x_0 + 0.5)^2 + (0.5x_1 - 0.83)^2 .$$

The results of the multifidelity optimization calculation are displayed in Table 4.4 and Figure 4.4. The results show that the space mapping parameters are nearly an exact match regardless of the number of high fidelity responses used by the inner loop. There is a sign

difference in some of the mapping parameters, but this is still a match since the terms are squared for use in the algorithm. A beneficial speed-up over traditional APPSPACK is also evident.

4.1.5 Polynomial Test 5: $\alpha \sim O(10), \gamma \sim O(100)$; $\beta = 1$

The high fidelity function with the exact form

$$f_H(x_0, x_1) = (8.0x_0 + 50.0)^2 + (5.0x_1 - 83.0)^2$$

is used in polynomial test case 5. The comparison between the APPSPACK and MFO results are shown in Table 4.5 and Figure 4.5. As was the case for the previous tests, the space mapping parameters are nearly an exact match for all the cases and scale well with the polynomial test case 1. Although there is a sign difference for some of the mapping parameters, the terms are squared when used so this can still be considered a match.

4.1.6 Polynomial Test 6: $\alpha \sim O(100), \gamma \sim O(10000)$; $\beta = 1$

For the final polynomial test, the high fidelity function has the exact form

$$f_H(x_0, x_1) = (80.0x_0 + 5000.0)^2 + (50.0x_1 - 8300.0)^2 .$$

Table 4.6 and Figure 4.6 display the results obtained using both the MFO and the traditional APPSPACK algorithms. The space mapping parameters are nearly an exact match and a significant speed-up is observed.

# Hi-Fi Resp.	γ_0 0.5	γ_1 -0.83	x_0 -0.5	x_1 0.83	Objective Value	# Hi-Fi Calc.	X Speed Imp
2	0.50	-0.83	-0.499	0.835	2.86e-5	21	2.76
4	0.50	-0.83	-0.50	0.83	1.0e-20	25	2.32
6	0.50	-0.83	-0.50	0.83	2.0e-20	28	2.07
8	0.50	-0.83	-0.50	0.83	4.0e-20	32	1.81
APPS	—	—	-0.50	0.85	4.0e-4	58	—

Table 4.1. Polynomial test case 1: $f_H(x_0,x_1) = (x_0+0.5)^2 + (x_1-0.83)^2$ and $f_L = y_0^2 + y_1^2$ with starting point $(x_0,x_1) = (-2.0,-2.0)$. Note the exact matching of the mapping parameters.

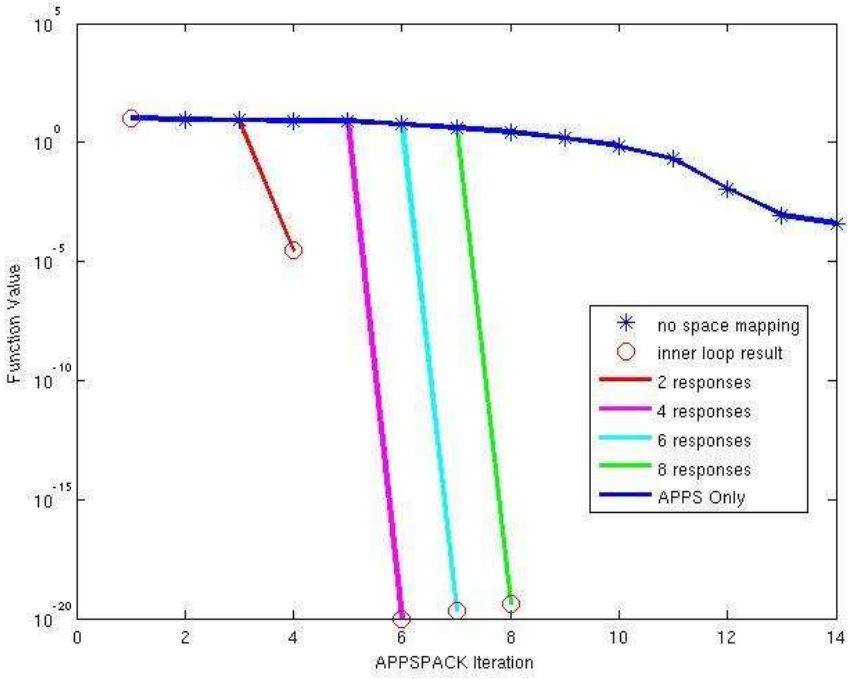

Figure 4.1. Polynomial Test Case 1: $f_H(x_0,x_1) = (x_0+0.5)^2 + (x_1-0.83)^2$ and $f_L = y_0^2 + y_1^2$. In searching for a minimum of f_H, the MFO algorithm progresses much quicker than traditional APPSPACK.

# Hi-Fi Resp.	γ_0 5.0	γ_1 -8.3	x_0 -5.0	x_1 8.3	Objective Value	# Hi-Fi Calc.	X Speed Imp
2	5.0	-8.3	-5.0	8.3	1.3e-17	21	2.76
4	5.0	-8.3	-5.0	8.3	1.0e-18	25	2.32
6	5.0	-8.3	-5.0	8.3	1.7e-17	28	2.07
8	5.0	-8.3	-5.0	8.3	0.0	32	1.81
APPS	—	—	-5.0	8.5	4.0e-2	58	—

Table 4.2. Polynomial Test Case 2: $f_H(x_0,x_1) = (x_0+5.0)^2 + (x_1-8.3)^2$ and $f_L = y_0^2 + y_1^2$ with starting point $(x_0,x_1) = (-20.0,-20.0)$. Note the exact matching of the mapping parameters and the nearly identical scaling to polynomial test case 1.

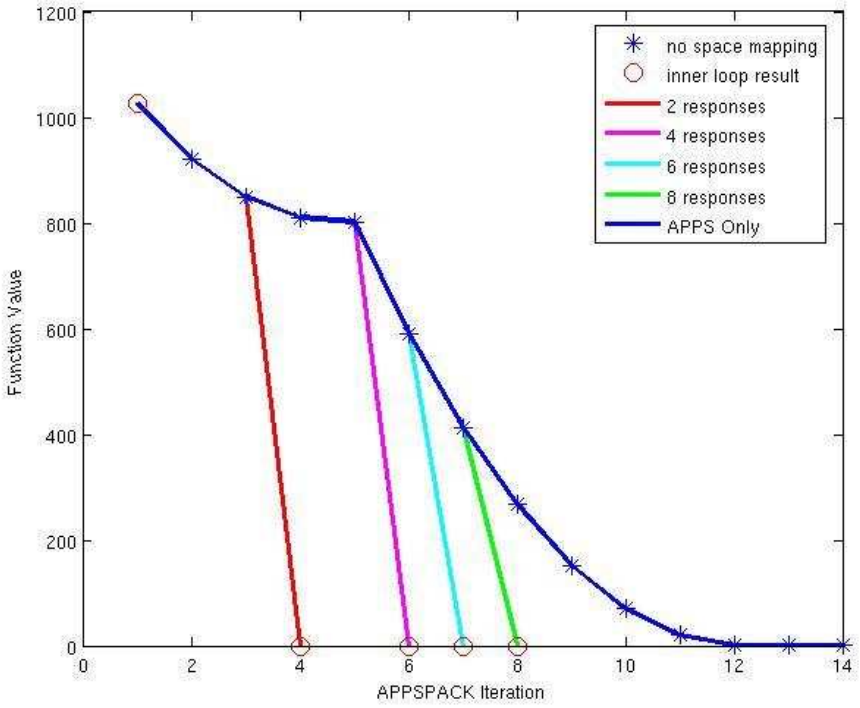

Figure 4.2. Polynomial Test Case 2: $f_H(x_0,x_1) = (x_0+5.0)^2 + (x_1-8.3)^2$ and $f_L = y_0^2 + y_1^2$. The MFO method provides improved algorithm progression and results. Note that this is a linear plot since the "8 response" case is exactly 0.

# Hi-Fi Resp.	γ_0 50.0	γ_1 -83.0	x_0 -50.0	x_1 83.0	Objective Value	# Hi-Fi Calc.	X Speed Imp
2	50.0	-83.0	-50.0	82.7	6.74e-2	21	2.76
4	50.0	-83.0	-50.0	83.0	1.0e-16	25	2.32
6	50.0	-83.0	-50.0	83.0	0.0	28	2.07
8	50.0	-83.0	-50.0	83.0	0.0	32	1.81
APPS	—	—	-50.0	85.0	4.0e-2	58	—

Table 4.3. Polynomial Test Case 3: $f_H(x_0,x_1) = (x_0+50.0)^2 + (x_1-83.0)^2$ and $f_L = y_0^2 + y_1^2$ with starting point $(x_0,x_1) = (-200.0,-200.0)$. Note the exact matching of the mapping parameters and the nearly identical scaling to polynomial test cases 1 and 2.

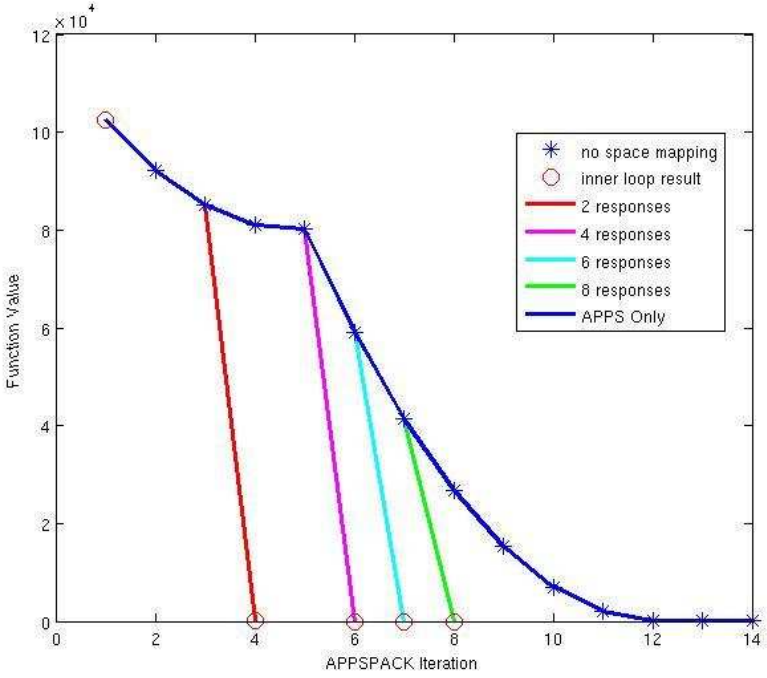

Figure 4.3. Polynomial Test Case 3: $f_H(x_0,x_1) = (x_0+50.0)^2 + (x_1-83.0)^2$ and $f_L = y_0^2 + y_1^2$. The minimum obtained by the MFO method is better than the one found by traditional APPSPACK. Note that this is a linear plot since the "6 response" and "8 responses" cases is exactly 0.

# Hi-Fi Resp.	α_0 0.8	α_1 0.5	γ_0 0.5	γ_1 -0.83	x_0 -0.625	x_1 1.66	Objective Value	# Hi-Fi Calc.	X Speed Imp
4	-0.80	0.50	-0.50	-0.83	-0.625	1.66	1.36e-15	26	2.73
6	0.80	0.50	0.50	-0.83	-0.625	1.66	9.35e-16	30	2.37
8	0.80	0.50	0.50	-0.83	-0.625	1.66	5.76e-16	30	2.37
APPS	—	—	—	—	-0.64	1.65	1.69e-4	71	—

Table 4.4. Polynomial Test Case 4: $f_H(x_0,x_1) = (0.8x_0+0.5)^2 + (0.5x_1-0.83)^2$ and $f_L = y_0^2 + y_1^2$ with starting point $(x_0,x_1) = (-2.0,-2.0)$. Note the exact matching of the mapping parameters and speed-up from APPSPACK.

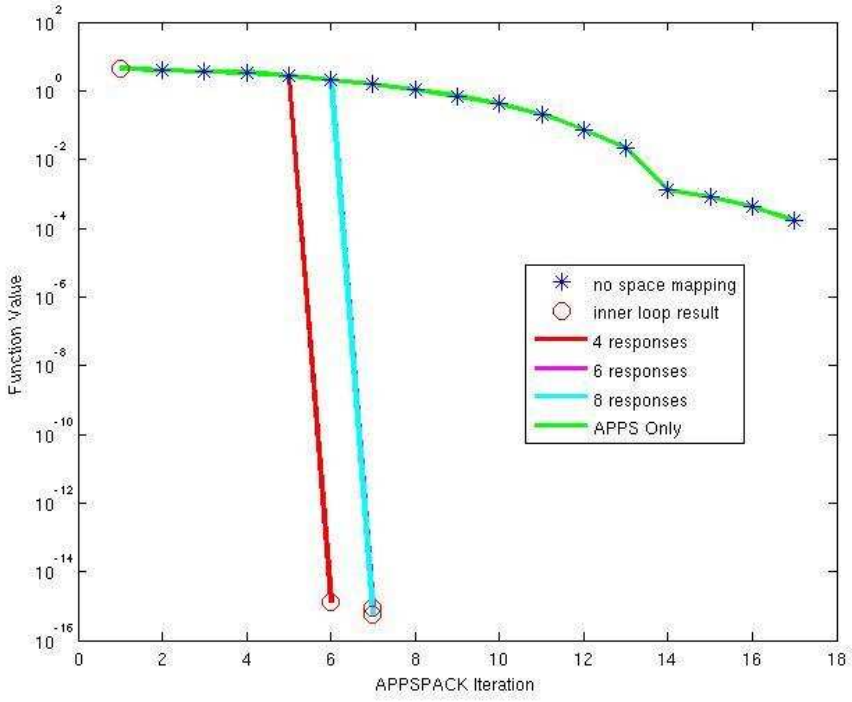

Figure 4.4. Polynomial Test Case 4: $f_H(x_0,x_1) = (0.8x_0+0.5)^2 + (0.5x_1-0.83)^2$ and $f_L = y_0^2 + y_1^2$. The addition of the inner loop in the MFO method improves progression.

# Hi-Fi Resp.	α_0 8.0	α_1 5.0	γ_0 50.0	γ_1 -83.0	x_0 -6.25	x_1 16.6	Objective Value	# Hi-Fi Calc.	X Speed Imp
4	-8.0	5.0	-50.0	-83.0	-6.25	16.6	4.06e-14	26	2.69
6	-8.0	5.0	-50.0	-83.0	-6.25	16.6	6.4e-17	30	2.33
8	8.0	5.0	50.0	-83.0	-6.25	16.6	4.1e-15	30	2.33
APPS	—	—	—	—	-6.4	16.5	1.69	70	—

Table 4.5. Polynomial Test Case 5: $f_H(x_0,x_1) = (8.0x_0+50.0)^2+(5.0x_1-83.0)^2$ and $f_L = y_0^2+y_1^2$ with starting point $(x_0,x_1) = (-20.0,-20.0)$. Note the exact matching of the mapping parameters and the scaling with the polynomial in test case 1.

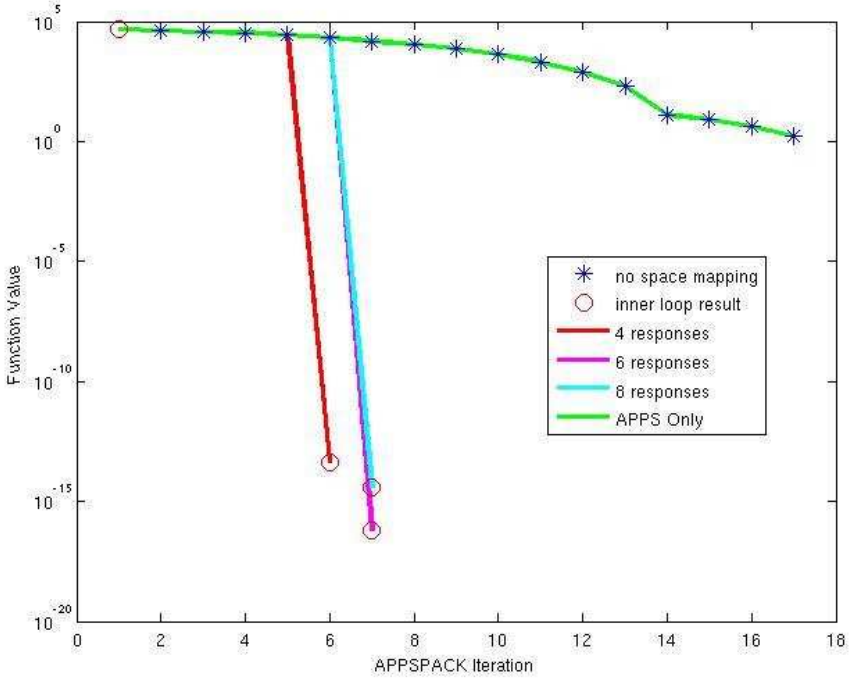

Figure 4.5. Polynomial Test Case 5: $f_H(x_0,x_1) = (8.0x_0+50.0)^2+(5.0x_1-83.0)^2$ and $f_L = y_0^2+y_1^2$. Note the improvement of MFO over APPSPACK.

# Hi-Fi Resp.	α_0 80.0	α_1 50.0	γ_0 5000.0	γ_1 -8300.0	x_0 -62.5	x_1 166.0	Objective Value	# Hi-Fi Calc.	X Speed Imp
4	80.0	50.0	5000.0	-8300.0	-62.5	166.0	2.81e-10	26	2.69
6	80.0	50.0	5000.0	-8300.0	-62.5	166.0	3.14e-11	30	2.33
8	80.0	50.0	5000.0	-8300.0	-62.5	166.0	3.08e-11	30	2.33
APPS	—	-	—	—	-6.4	16.5	1.69e+4	70	—

Table 4.6. Polynomial Test Case 6: $f_H(x_0,x_1) = (80.0x_0 + 5000.0)^2 + (50.0x_1 - 8300.0)^2$ and $f_L = y_0^2 + y_1^2$ with starting point $(x_0,x_1) = (-200.0,-200.0)$. The space mapping parameters are almost an exact match.

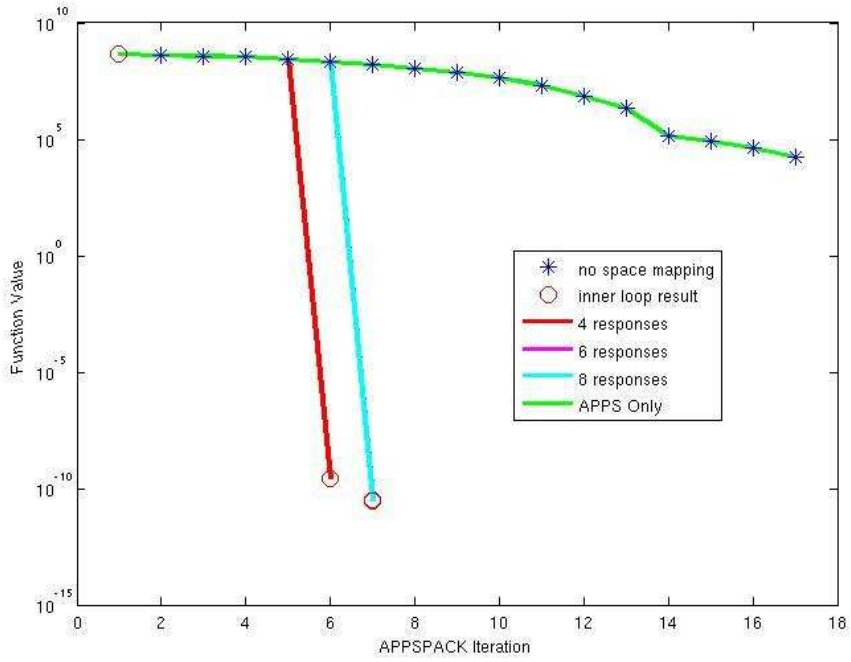

Figure 4.6. Polynomial Test Case 6: $f_H(x_0,x_1) = (80.0x_0 + 5000.0)^2 + (50.0x_1 - 8300.0)^2$ and $f_L = y_0^2 + y_1^2$. The speed-up of the MFO algorithm over APPSPACK is about 2 times.

4.2 Multi-Variable Rosenbrock Function

In testing optimization methods, it is common to study the feasibility of an algorithm on the Rosenbrock function. The function is

$$f_R = 100\left(y_1 - y_0^2\right)^2 + (1 - y_0)^2, \tag{4.6}$$

and has a minimum at $f_R(1,1) = 0.0$. The multi-variable Rosenbrock function has N design variables and can be written as

$$f_{MVR}(x_0, \cdots, x_{N-1}) = \sum_{k=1}^{N-1} \left\{ 100\left(x_k - x_{k-1}^2\right)^2 + \left(1 - x_{k-1}^2\right)^2 \right\}, \tag{4.7}$$

where $f_{MVR}(1, \cdots, 1) = 0.0$ is a minimum. Note that (4.7) for $N = 2$ is merely (4.6). Thus, the Rosenbrock equation can be used as a low fidelity model and the multi-variable Rosenbrock as a corresponding high fidelity model. We examine these models to study the feasibility of mapping between models with different numbers of design variables.

To map between models with different numbers of design variables, a general mapping was developed that captures the cross-dependencies of mapping parameters. This general mapping has the following form

$$P^{(n)} = A^{(m,n)} x_H^{(n)} + \Gamma^{(m)} \tag{4.8}$$

where

$$A^{(m,n)} = \begin{bmatrix} \alpha_{00} & \cdots & \alpha_{0n} \\ \vdots & \ddots & \vdots \\ \alpha_{m0} & \cdots & \alpha_{mn} \end{bmatrix}, \Gamma^{(m)} = \begin{bmatrix} \gamma_0 \\ \vdots \\ \gamma_m \end{bmatrix}, \tag{4.9}$$

m is the number of low fidelity variables, and n is the number of high fidelity variables. The general mapping (4.8) can be reduced to our previous mapping, $m = n$, by using only the diagonal terms of $A^{(m,n)}$. Note that when the cross terms are required, additional high fidelity response calculations are required.

The following test cases were analyzed using the standard Rosenbrock function as the low fidelity model and the multi-variable Rosenbrock as the high fidelity model:

Test 1: $N = 3$, $\alpha_{00}, \ldots, \alpha_{mn} \sim O(1)$, $\gamma_0, \ldots, \gamma_m = 0$

Test 2: $N = 4$, $\alpha_{00}, \ldots, \alpha_{mn} \sim O(1)$, $\gamma_0, \ldots, \gamma_m = 0$

Test 3: $N = 3$, $\alpha_{00}, \ldots, \alpha_{mn} \sim O(1)$, $\gamma_0, \ldots, \gamma_m \sim O(1)$

As with the polynomial test cases, the results from each test case are included in the next sections in both table and graph formats. The tables contain basically the same information as previously. One change is that the first column is now the the number of non-zero space mapping parameters, from matrix (4.9), used in the calculations (# SM parameters). The last row of the table (APPS) still displays results obtained when APPSPACK was applied to the high fidelity model (*e.g.* no calls to the inner loop). The other columns in the table include the final values of the design variables $(x_0, \cdots, x_N)$, the final objective function value (Objective Value), the total number of high fidelity response calculations completed (# Hi-Fi Resp.), and the speed improvement of the MFO algorithm from traditional APPSPACK (X Speed Imp).

As before, the graph illustrates the algorithmic progression of each row in the table. The x-axis indicates the number of successful APPSPACK iterations where successful refers to identifying of a new best point. The y-axis shows the value of the objective function. The legend indicates which color corresponds to what number of non-zero high fidelity space mapping parameters were used. The circles and stars indicate whether or not an inner loop calculation was done to obtain the point respectively.

4.2.1 Multi-Variable Rosenbrock Test 1: $N = 3$, $\alpha_{00}, \ldots, \alpha_{mn} \sim O(1)$, $\gamma_0, \ldots, \gamma_m = 0$

This test uses the high fidelity function of the exact form

$$f_H(x_0, x_1, x_2) = 100\left(x_1 - x_0^2\right)^2 + (1 - x_0)^2 + 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2, \tag{4.10}$$

and the exact mapping terms

$$A^{(2,3)} = \begin{bmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} \end{bmatrix}, \Gamma^{(2)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{4.11}$$

In Table 4.7, results given the following testing conditions are displayed:

- 3 non-zero space mapping parameters; $\alpha_{10} = \alpha_{01} = \alpha_{02} = 0.0$
- 4 non-zero space mapping parameters; $\alpha_{10} = \alpha_{01} = 0.0$
- 6 non-zero space mapping parameters; all α are non-zero

Since the multi-variable Rosenbrock is a sum of squares whose minimum is 0, the equation can be solved by setting each individual squared term equal to zero. Then, (4.10) shows

that x_0 and x_1 are forced to their optimum with the $(1-x_0)^2$ and $(1-x_1)^2$ respectively. Therefore x_2 is only dependent on the x_1 term via $\left(x_2-x_1^2\right)^2$, and the only relevant cross term is α_{12}. Basically, the closer the problem is to the objective function the "non-relevant" space mapping parameters will go to zero.

Both Table 4.7 and Figure 4.7 show that a solution is found the quickest in the the case of using three space mapping parameters. However, if four space mapping parameters are used, the speed-up is still significant and a better solution can be obtained.

4.2.2 Multi-Variable Rosenbrock Test 2: $N = 4$, $\alpha_{00},\ldots,\alpha_{mn} \sim O(1)$, $\gamma_0,\ldots,\gamma_m = 0$

In test 2, the high fidelity function has the exact form

$$f_H(x_0,x_1,x_2) = 100\left(x_1-x_0^2\right)^2+(1-x_0)^2+100\left(x_2-x_1^2\right)^2+(1-x_1)^2 \\ +100\left(x_3-x_2^2\right)^2+(1-x_2)^2, \tag{4.12}$$

and the exact mapping terms

$$A^{(2,3)} = \begin{bmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \end{bmatrix}, \Gamma^{(2)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{4.13}$$

Table 4.8 includes results for the following testing conditions:

- 4 non-zero space mapping parameters; $\alpha_{10} = \alpha_{01} = \alpha_{02} = \alpha_{03} = 0.0$
- 8 non-zero space mapping parameters; all α are non-zero

As illustrated in Table 4.8 and Figure 4.8, the case with four non-zero space mapping parameters gives the best answer with the least amount of computational effort.

4.2.3 Multi-Variable Rosenbrock Test 3: $N = 3$, $\alpha_{00},\ldots,\alpha_{mn} \sim O(1)$, $\gamma_0,\ldots,\gamma_m \sim O(1)$

The exact form high fidelity function used in test 3 is

$$f_H(x_0,x_1,x_2) = 100\left(x_1-x_0^2\right)^2+(1-x_0)^2+100\left(x_2-x_1^2\right)^2+(1-x_1)^2, \tag{4.14}$$

and the exact mapping terms are

$$A^{(2,3)} = \begin{bmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} \end{bmatrix}, \Gamma^{(2)} = \begin{bmatrix} \gamma_0 \\ \gamma_1 \end{bmatrix}. \tag{4.15}$$

Table 4.9 giving results for the following test conditions:

- 5 space parameters - $\alpha_{10} = \alpha_{01} = \alpha_{02} = 0.0$ and all γ are non-zero
- 6 space parameters - $\alpha_{10} = \alpha_{01} = 0.0$ and all γ are non-zero
- 8 space parameters - all α and γ are non-zero

The results are illustrated in Figure 4.9. The test condition with eight non-zero space mapping parameters gives a significantly better solution. It requires more computational work than the other MFO runs, but it still provides some improvement over traditional APPSPACK.

# SM Parameters	x_0 1.0	x_1 1.0	$x2$ 1.0	Objective Value	# Hi-Fi Calc.	X Speed Imp
3	7.29e-1	5.45e-1	2.93e-1	3.04e-1	42	2.98
4	8.29e-1	6.80e-1	4.57e-1	1.39e-1	50	2.50
6	2.99e-1	6.8e-1	4.57e-1	1.35	87	1.44
APPS	3.50e-1	1.20e-1	-1.10e-16	1.218	125	—

Table 4.7. Multi-Variable Rosenbrock Test Case 1: $f_H = 100\left(x_1 - x_0^2\right)^2 + (1-x_0)^2 + 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2$ and $f_L = 100\left(y_1 - y_0^2\right)^2 + (1-y_0)^2$.

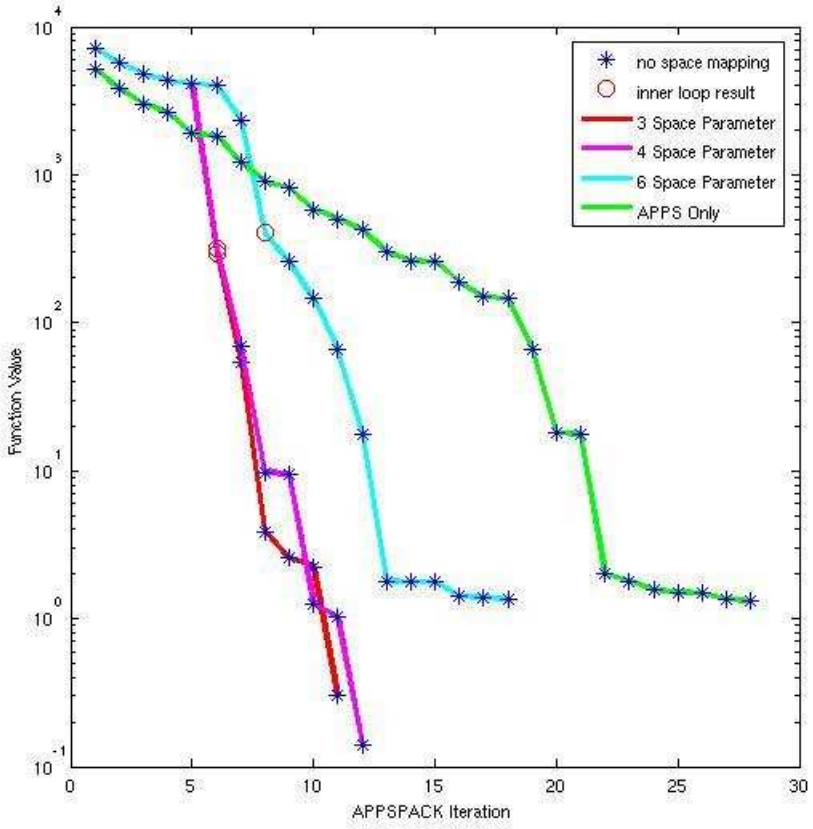

Figure 4.7. Multi-Variable Rosenbrock Test Case 1: $f_H = 100\left(x_1 - x_0^2\right)^2 + (1-x_0)^2 + 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2$ and $f_L = 100\left(y_1 - y_0^2\right)^2 + (1-y_0)^2$.

# SM Parameters	x_0 1.0	x_1 1.0	$x2$ 1.0	x_3 1.0	Objective Value	# Hi-Fi Calc.	X Speed Imp
4	4.94e-1	2.41e-1	8.12e-2	9.02e-3	1.73	80	2.36
8	5.54e-1	2.89e-1	8.69e-2	-3.15e-3	1.58	154	1.23
APPS	3.20e-1	9.00e-2	-4.00e-2	-1.10e-16	2.312	189	—

Table 4.8. Multi-Variable Rosenbrock Test Case 2: $f_H = 100\left(x_1 - x_0^2\right)^2 + (1-x_0)^2 + 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2 + 100\left(x_3 - x_2^2\right)^2 + (1-x_2)^2$ and $f_L = 100\left(y_1 - y_0^2\right)^2 + (1-y_0)^2$

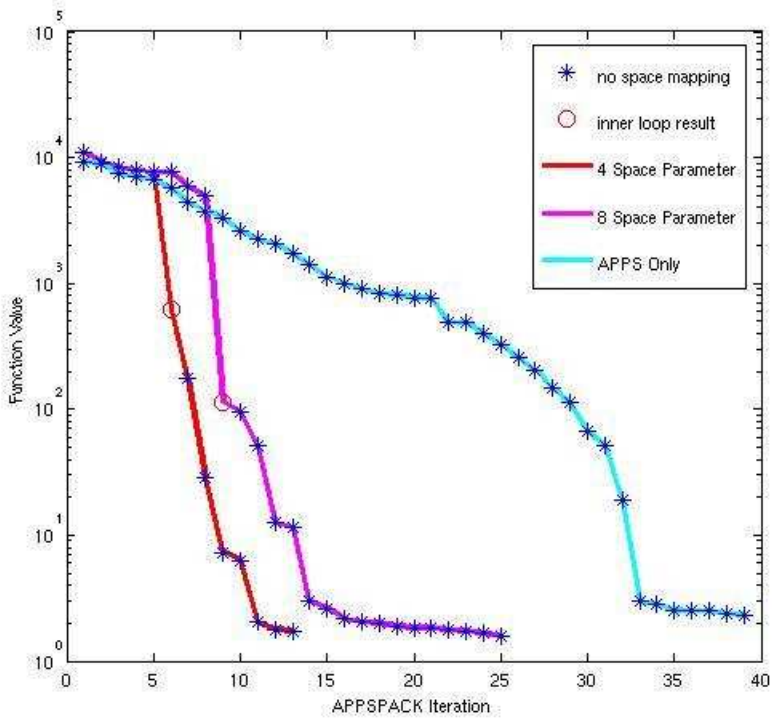

Figure 4.8. Multi-Variable Rosenbrock Test Case 2: $f_H = 100\left(x_1 - x_0^2\right)^2 + (1-x_0)^2 + 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2 + 100\left(x_3 - x_2^2\right)^2 + (1-x_2)^2$ and $f_L = 100\left(y_1 - y_0^2\right)^2 + (1-y_0)^2$.

# SM Parameters	x_0 1.0	x_1 1.0	$x2$ 1.0	Objective Value	# Hi-Fi Calc.	X Speed Imp
5	5.48e-1	3.18e-1	1.18e-1	7.28e-1	50	2.50
6	3.48e-1	1.21e-1	6.56e-3	1.20	62	2.02
8	9.50e-1	9.06e-1	8.35e-1	3.18e-2	91	1.38
APPS	3.50e-1	1.20e-1	-1.10e-16	1.218	125	—

Table 4.9. Multi-Variable Rosenbrock Test Case 3: $f_H = 100\left(x_1 - x_0^2\right)^2 + (1-x_0)^2 + 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2$ and $f_L = 100\left(y_1 - y_0^2\right)^2 + (1-y_0)^2$.

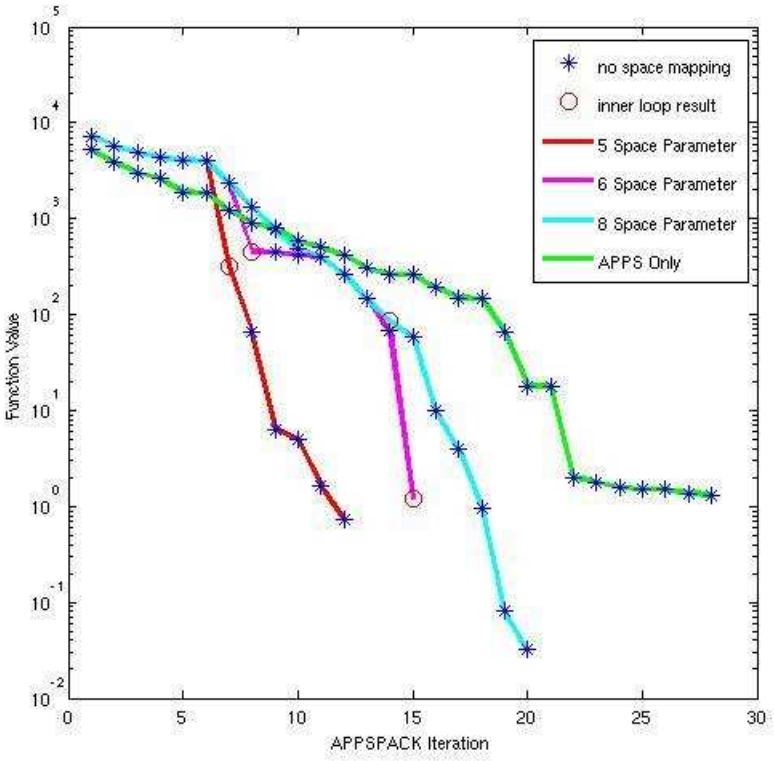

Figure 4.9. Multi-Variable Rosenbrock Test Case 3: $f_H = 100\left(x_1 - x_0^2\right)^2 + (1-x_0)^2 + 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2$ and $f_L = 100\left(y_1 - y_0^2\right)^2 + (1-y_0)^2$.

Chapter 5

Engineering Science Application

Although our MFO algorithm proved to be promising for the test problems, we wanted to demonstrate its usefulness on an engineering sciences application. We began by considering earth penetrators. Our goal was to work with the earth penetrator design community to help solve an important problem. However, we met with some resistance due to the politics of earth penetrators. For example, due to cross-laboratory competition of earth penetrator designs, developers were reluctant to share their models with researchers. In addition, earth penetrators fell out of favor in Congress and received significant funding cuts in FY04. Therefore, we switched our focus to an entirely different application, the groundwater remediation problem. We begin this section describing the preliminary, yet promising, results for an earth penetrator problem. We end with a overview of the groundwater remediation problem and our complete results.

5.1 Earth Penetrator Models

Earth penetrators models consist of a deformable projectile subjected to extremely high strain rates. This is typically a missile like projectile that may be composed of multiple materials making impact with a solid target that may be composed of layers of material including concrete, soft soil, hard soil, etc. Typically, the input parameters in such a model are the penetrator impact velocity, angle of attack, material composition, and target material composition (see Figure 5.1). The model response values of interest include the penetrator's displacement within the target and the penetrator's acceleration at impact.

The Sandia code PRONTO 3D [33, 36, 7] was used to implement earth penetrator models. PRONTO 3D is a three-dimensional transient solid dynamics code for analyzing large deformations of highly nonlinear materials subjected to extremely high strain rates. The penetrator may be modeled in two modes: (i) compressible mode to allow for deformation of the penetrator as it impacts the target and (ii) rigid mode so that no deformation occurs during impact. The compute time is significantly different for these two modes. An 8000

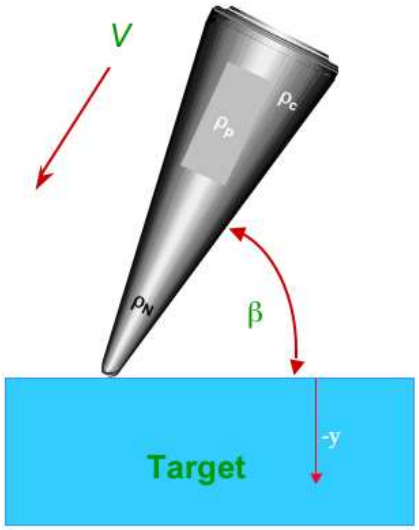

Figure 5.1. Simple graphic depicting an earth penetrator composed of multiple materials (depicted by density ρ) striking a target.

element mesh in the compressible mode takes approximately 40 minutes to execute while the rigid mode takes only about 10 seconds on the same computer, a computational ratio of 1:240. A parameter study varying the penetrator density and corresponding acceleration response is shown in Figure 5.2. The elastic body results in an oscillatory acceleration response since the body is compressible and pressure waves within the body produce these noisy oscillations. In contrast, the rigid body has a relatively smooth acceleration response with respect to density, but it captures the trend of the elastic body. This factor and the difference in compute times make this problem an ideal candidate for multifidelity optimization.

Many different earth penetrator calculations were done minimizing acceleration and maximizing displacement by varying penetrator density. With these calculations, a 2-3 times speed up was observed with displacement and 1-2 times speed was observed with acceleration. One case of particular interest is shown in Figure 5.3 and Table 5.1. The acceleration of the penetrator was minimized such that

- high fidelity model = elastic model with fine mesh (see Figure 5.2) ,

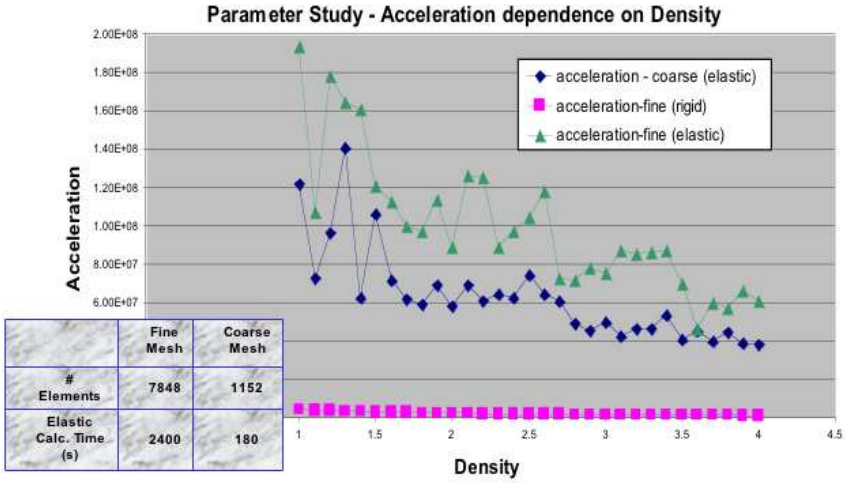

Figure 5.2. Parameter study of acceleration response of earth penetrator with varying density impacting a concrete target. Note how the rigid penetrator captures the trend of the elastic penetrator without the ”noise”.

- low fidelity model = rigid model with fine mesh.

As Figure 5.3 illustrates, the MFO approach actually takes longer, but finds an improved minimum. The reason for this is that traditional APPSPACK becomes trapped at a local minimum. Since the acceleration response of the rigid body is smooth and captures the correct trend, it pushes the optimization past the local noise and finds the a better minimum (see Figure 5.4). This result is in no way implying that our MFO algorithm guarantees a global minimum, just that the behavior was observed in this specific case.

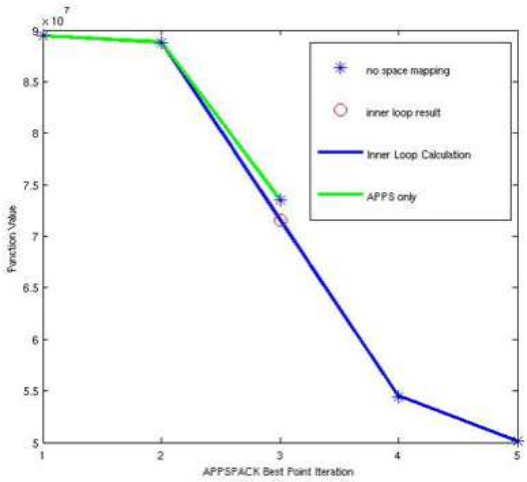

Figure 5.3. Comparison of traditional APPSPACK and MFO (specified as Inner Loop Calculation) on the problem of minimizing the acceleration of an earth penetrator. See Table 5.1 for specifics. Although the MFO case takes longer to optimize, it finds the "observed" optimum (see Figure 5.4).

Optimization Scheme	Objective Value	# Hi-Fi Calc.	X Speed Imp
MFO	5.0e7	11	0.82
APPS	7.3e7	9	—

Table 5.1. Comparison of traditional APPSPACK and MFO on the problem of minimizing acceleration of an earth penetrator. Although the MFO case takes longer to optimize, it finds a better optimum (see Figure 5.4).

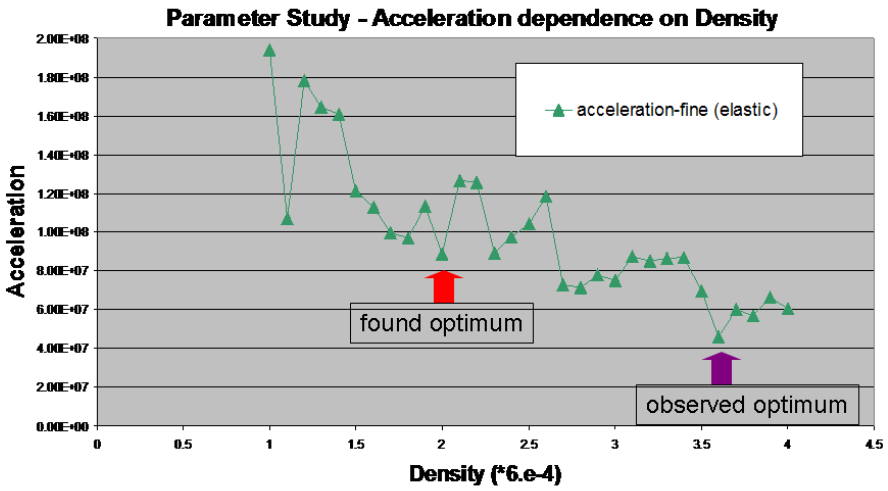

Figure 5.4. Traditional APPSPACK calculates the local optimum specified with the red arrow. The MFO scheme demonstrates a global search capability and finds a better optimum at the purple arrow. From the parameter study this is the observed optimum, but not necessarily the global optimum.

5.2 Groundwater Remediation Models

Remediation is a technique that is employed to reduce or minimize the negative health and environmental impacts of chemical contamination. One method of groundwater remediation, hydraulic capture (HC), uses the design of a well field to alter the direction of the groundwater flow and reverse or halt the migration of the contaminant plume. For example, a well field can be designed so that gradients point toward the interior of the plume, resulting in containment and possible shrinkage of the plume. The overall goal of HC is to contain or shrink a plume at minimum cost. Therefore, optimization is a useful tool for the design of an appropriate well field.

Plume migration control is an active area of research in environmental engineering. The three prime areas of focus are: improving models of subsurface flow and transport, incorporating installation and operational costs of HC into the objective functions, and developing and calibrating capture constraints so that computational efficiency is obtained in the modeling process while simultaneously avoiding excessive pumping in the final well design. Once an objective function and constraints are developed, optimization techniques are used to determine the well design that minimizes the remediation cost while satisfying the constraints. Mathematically, these problems are challenging because evaluation of the objective function and constraints typically requires the results of a simulation. This means that necessary gradient information may be difficult or impossible to obtain.

The problem studied here is motivated by a hydraulic capture application proposed as part of a suite of benchmarking test problems in [27]. This entire test suite of problems includes data for several different physical domains and objective functions as well as guidelines for comparison. For the HC application, it is left to the modeler to choose the capture constraint and the concentration contour of the plume boundary.

5.2.1 Models of Hydraulic Capture

In this study, we consider two hydraulic capture models: transport-based concentration control (TBCC) and flow-based hydraulic control (FBHC). For both models, the optimization problem is

$$\min_{\mathbf{u}\in\Omega} J(\mathbf{u}) \tag{5.1}$$

where $\mathbf{u}$ is a vector of decision variables and Ω is the feasible region of $\mathbf{u}$ and can be represented by a set of constraint equations.

The objective function J is merely the sum of the capital (or installation) cost J^c and the operational cost J^o and can be stated as follows [27, 26]:

$$\begin{aligned} J &= \underbrace{\sum_{i=1}^{n} c_0 d_i^{b_0} + \sum_{Q_i<0.0} c_1 |Q_i^m|^{b_1} (z_{gs} - h^{min})^{b_2}}_{J^c} \\ &+ \underbrace{\int_0^{t_f} \left(\sum_{i,Q_i<0.0} c_2 Q_i (h_i - z_{gs}) + \sum_{i,Q_i>0.0} c_3 Q_i \right) dt}_{J^o}. \end{aligned} \tag{5.2}$$

In J^c, the first term accounts for drilling and installing all the wells and the second term represents the additional cost for pumps for extraction wells. In J^o, the term pertaining the extraction wells includes the lift cost associated with raising the water to the surface. More specifically, in (5.2) c_j and b_j are cost coefficients and exponents, respectively, $d_i = z_{gs}$ is the depth of well i, Q_i^m is the design pumping rate, and h^{min} is the minimum allowable head. We use the values $h^{min} = 10(m)$ and $Q_i^m = \pm 0.0064(m^3/s)$ and $d_i = 30(m)$ for each pump i. Note that a negative pumping rate means a well is extracting and a positive pumping rates means a well is injecting and that injection wells are assumed to operate under gravity feed conditions. The simulation time is $t_f = 5$ years. Other pertinent cost data is given in [26].

The number of wells, $n \leq N$, and pumping rates $\{Q_i\}_{i=1}^n$ are decision variables. If, in the course of the optimization, a well rate satisfies the inequality $|Q_i| \leq 10^{-6}(m^3/s)$, that well is removed from the design space and excluded from all other calculations. Although this restriction leads to a discontinuous objective function, the benefit is a decrease in the installation cost and a well design with fewer wells operating at higher rates. We assume that all wells have a fixed depth, but that their location can vary in the $x-y$ plane. These well locations, $\{(x_i, y_i)\}_{i=1}^n$ are also decision variables, but do not explicitly appear in the objective function.

The hydraulic heads, $h_i(m)$ for well i, also vary with the decision variables and obtaining their values at each iteration requires a solution to equations that model saturated flow. The model is given by

$$S_s \frac{\partial h}{\partial t} = \nabla \cdot (\mathbf{K} \cdot \nabla h) + \bar{S}, \tag{5.3}$$

where $S_s(1/m)$ is the specific storage coefficient, $h(m)$ is the hydraulic head, $\mathbf{K}(m/s)$ is the hydraulic conductivity tensor, and $\bar{S}(m^3/s)$ is a fluid source term and is where the decision variables enter into the state equation for the HC problem. Numerically, the simulator MODFLOW2000 [38] is used to find a solution to (5.3).

In HC models, constraints on the decision variables typically include bounds on the well capacities and the hydraulic head at each well location. For example, we incorporate

the inequalities

$$Q^{emax} \leq Q_i \leq Q^{imax}(m^3/s), i = 1,...,n \tag{5.4}$$

$$h^{max} \geq h_i \geq h^{min}(m), i = 1,...,n \tag{5.5}$$

where, Q^{emax} is the maximum extraction rate at any well, Q^{imax} is the maximum injection rate at any well, h^{max} is the maximum allowable head, and h^{min} is the minimum allowable head. Note that assessing whether or not (5.5) holds requires a solution to (5.3). In addition to (5.4) and (5.5), the HC problem constrains the net pumping rate using the inequality

$$Q_T = \sum_{i=1}^{n} Q_i \geq Q_T^{max}, \tag{5.6}$$

where $Q_T^{max}(m^3/s)$ is the maximum allowable total extraction rate. This ensures that the land is not completely drained of all its water resources.

5.2.1.1 Flow-based Hydraulic Control (FBHC)

Flow-based hydraulic control is a technique for plume containment that enforces head gradient constraints around the perimeter of the plume. In the particular model we consider, a head gradient constraint is formulated as a constraint on the difference in hydraulic head values at specified locations. Consider

$$h_1^k - h_2^k \geq d(m), k = 1 \ldots M \tag{5.7}$$

where M is the number of head gradient constraints imposed around the boundary, h_1, h_2 are hydraulic head values at specified nodes for each constraint k, and d is the bound on the difference. This set of constraints can be used to enforce head gradients vertically or horizontally. For example, in the simple case where h_1 are h_2 are aligned at a distance of distance Δx apart, then dividing (5.7) by Δx yields

$$\left(\frac{h_1^k - h_2^k}{\Delta x}\right) \geq \frac{d}{\Delta x}(m/s). \tag{5.8}$$

The FBHC approach to the HC problem is minimizing the objective function J subject to (5.4), (5.5), (5.6) and (5.7). Use of this method is attractive because it is relatively inexpensive. However, it requires (5.7) to be calibrated to ensure that the contaminant plume is properly captured and to avoid excessive pumping [28].

5.2.1.2 Transport Based Concentration Control (TBCC)

A direct approach for plume containment is to impose constraints on the concentration at specified locations. This constraint can be expressed as

$$C_j \leq C_j^{max}(kg/m^3) \tag{5.9}$$

where C_j is the concentration at some observation node j and C_j^{max} is the maximum allowable concentration. Evaluation of this constraint requires a solution to the contaminant transport equation

$$\frac{\partial(\theta^{\alpha}C^{\iota})}{\partial t} = \nabla\cdot(\theta^{\alpha}\mathbf{D}^{\iota}\cdot\nabla C^{\iota}) - \nabla\cdot(\mathbf{q}C^{\iota}) + I^{\iota} + R^{\iota} + S^{\iota}, \tag{5.10}$$

where $C^{\iota}(kg/m^3)$ is the concentration of species ι in the aqueous phase, θ^{α} is the volume fraction of the aqueous phase, $\mathbf{D}$ is a hydrodynamic dispersion tensor, $\mathbf{v}(m/s)$ is the mean pore velocity, $\mathbf{q}(m/s)$ is the Darcy velocity, and $I^{\iota}, R^{\iota}, S^{\iota}$ represent interphase mass transfer, biogeochemical reactions, and source of mass respectively. Numerically, the simulator known as MT3DMS [39] is used to obtain a solution to the transport equation.

Solving the HC problem using the TBCC model is minimizing the cost function J with respect to (5.4), (5.5), (5.6), and (5.9). Therefore, the TBCC approach requires a solution to both (5.10) and (5.3) making it computationally expensive. Moreover, objective functions and constraints involving concentrations are nonconvex in some situations [6, 3], making the minimization problem more difficult.

5.2.2 HC Problem for Comparison

In the MFO approach to the HC problem, we use the FBHC model as the low fidelity model and the TBCC model as the corresponding high fidelity model. In order to test the effectiveness of the MFO approach, we compare results for the HC problem included in the set of community problems proposed in [27]. For the simple domain of this problem, the FBHC formulation has been shown to be sufficient [12]. However, it should be noted that other approaches are needed for more realistic domains [1].

In the HC problem used here, the physical domain is a $1000\times1000\times30(m)$ unconfined aquifer. Since the aquifer is unconfined, the head constraint (5.5), depends nonlinearly on the pumping rates. For the hydraulic conductivity field, we use the simple homogeneous case with $K = 5.01\times10^{-5}(m/s)$. Paired with the saturated flow equation (5.3), we use the following boundary and initial conditions:

$$\left.\frac{\partial h}{\partial x}\right|_{x=0} = \left.\frac{\partial h}{\partial y}\right|_{y=0} = \left.\frac{\partial h}{\partial z}\right|_{z=0} = 0, \quad t > 0$$

$$q_z(x,y,z=h,t>0) = -1.903\text{x}10^{-8}(m/s), \text{ where } q_z = -K\frac{\partial h}{\partial z},$$
$$h(1000,y,z,t>0) = 20-0.001y(m)$$
$$h(x,1000,z,t>0) = 20-0.001x(m).$$

Here q_z is the Darcy flux out of the domain, representing recharge into the aquifer that could result from rainfall. The steady state solution to the flow problem without wells is $h(x,y,z,0) = h_s(m)$, and $S_s = 2.0\times10^{-1}(1/m)$ is the specific yield of the unconfined aquifer. The ground surface elevation is $z_{gs} = 30(m)$.

A plume development was simulated using (5.10) from a finite source for five years with a constant concentration of $1kg/m^3$ located in the region bounded by

$$[(200,225);(475,525),(h,h-2)](m).$$

We chose the $5\times10^{-5}(kg/m^3)$ contour line as the plume boundary and set $C_j^{max} = 5\times10^{-5}(kg/m^3)$ in (5.9). The MT3DMS [39] was used to generate this initial contaminant plume, as described in [26].

For the FBHC approach, we use five head difference constraints (5.7) with $d = 10^{-4}$. Five concentration constraints (5.9) are used for the TBCC approach, enforced in the same locations as (5.7) in the FBHC approach. The starting point includes two extraction and two injection wells for a total of $N = 4$ candidate wells and initial pumping rates of $\pm Q_i = 0.0064(m^3/s)$.

5.2.3 Numerical Results

The financial cost of the remediation at the initial iterate is $J(u_0) = \$78,587$. Table 5.2 shows the minimum financial cost found and the %-decrease for each of the approaches–FBHC, TBCC, and MFO. Both the FBHC and TBCC models were solved using APPSPACK 4.0, the same software customized for the APPS/Space Mapping approach to MFO. Note that all three methods produce solutions with similar remediations costs and overall percentage decreases.

One way to view the computational performance of the algorithms is to consider the number of function evaluations needed to reach the optimal point. As with most simulation-based optimization problems, the majority of the computational cost in the function evaluation is the calls to the MODFLOW2000 and/or MT3DMS simulators. Note that if (5.6) is not satisfied, a flow simulation is not performed in any approach and likewise if (5.5) is not satisfied, then no transport simulation is performed for the TBCC or MFO approaches. The fourth and fifth columns in Table 5.2 show the number of times that MODFLOW2000 (mf2k) and MT3DMS (mt3d) were called for each approach. For this problem, a MODFLOW2000 simulation takes approximately 2 seconds, wall clock time and a MT3DMS simulation takes anywhere from 40 to 50 seconds wall clock time.

Figure 5.5 highlights the differences in the three approaches by illustrating their performances over the course of the algorithm's execution. All three approaches start off the same. However, once the MFO approach receives the results of an inner loop calculation, it is able to make a significant decrease in the overall cost more quickly.

5.2.4 Discussion

In [27], many of the existing challenges in optimal design of problems involving saturated flow and transport are addressed. The lack of representative problems to be used for the testing and comparison of methods is discussed, and a set of test problems is presented. Although the test problems were developed after an extensive literature search and are indicative of real world problems, the domain is still relatively simple. For this simple, homogeneous case considered in this work, the FBHC model is sufficient and computationally cost-effective [11]. However, the introduction of heterogeneities or other complexities will likely require the TBCC or other models in which the plume boundary is precisely defined. Since this may not be a viable alternative with respect to computational cost, we offer the MFO approach. We have shown that the MFO approach is a comparable method and thus may be a reasonable method for the solution of groundwater remediation problems. To further investigate its usefulness in this case, we plan to extend our study to consider more representative physical models and simulators and to incorporate real-site data.

Model	Cost	% Decrease	mf2k calls	mt3d calls
FBHC	$24,175	69.2%	117	0
TBCC	$20,362	74.1%	188	160
MFO	$22,428	71.5%	152	86

Table 5.2. Comparison of results from solving a groundwater remediation problem using three different models. The first column gives the model used in the problem statement while the remaining four columns give the corresponding performance information.

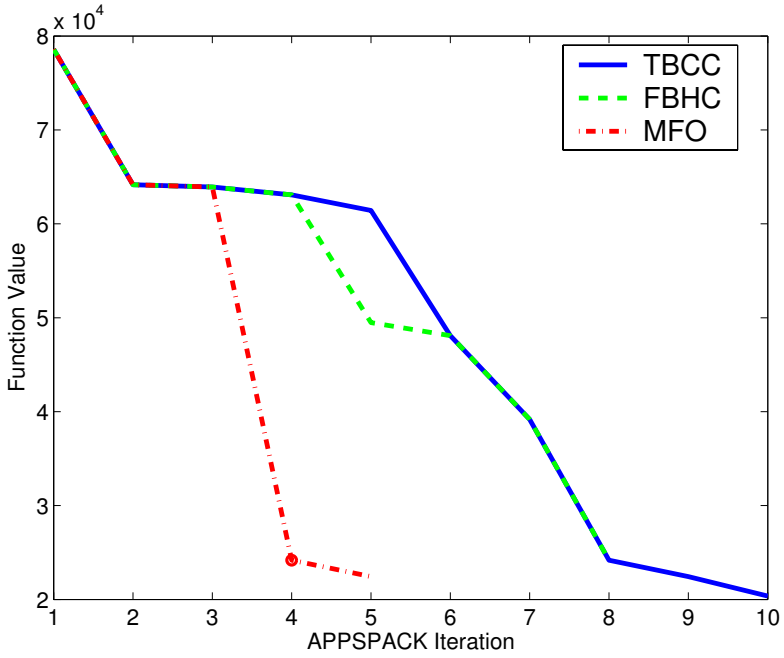

Figure 5.5. Graph showing the reduction in the groundwater remediation cost function versus the number of APPSPACK iterations. The solid blue line and green dashed lines correspond to the solutions of the TBCC and the FBHC models, respectively, solved with APPSPACK 4.0. The red dot-dash line corresponds to solving the problem using the MFO approach. Note that the significant decrease corresponding to the red dot (at iteration 4) was the result of and inner loop calculation.

Chapter 6

Conclusions and Future Work

In this paper, we have described an approach to multifidelity optimization based on asynchronous parallel pattern search and space mapping. Two important features of our approach are: (i) it is provably convergent and (ii) the number of design variables between models of differing fidelity may be non-equivalent in both number and type. Our MFO scheme was implemented using the existing APPSPACK software and a customized oracle. This oracle is composed of a series of scripts that control the calculation of the space mapped low fidelity optimization calculation. This report describes the implementation details and illustrates our success for a set of test problems including polynomial and multi-variable Rosenbrock and for two engineering science applications– earth penetrators and groundwater remediation.

Throughout the course of this work, some topics of interest arose that we were unfortunately unable to touch upon. Future work in this area should consider them. For example, a more generic and dynamic mapping could be developed through the use of data mining. In this case, the selection of the space mapping terms would be based on both past and runtime calculations, and the space mapping formulation would be improved at each run of the application. Another approach might include 1^{st} and 2^{nd} order corrective models in the mapping terms as done in [29]. Note that this would require intermittent calls to fixed patterns in order to calculate the derivate terms, and therefore, a more generic oracle implementation would be needed.

Currently, our MFO scheme is a set of standalone scripts that work in conjunction with APPSPACK. We believe it would be beneficial to incorporate our scheme into the DAKOTA toolkit. Future engineering science applications of our MFO algorithm include an extended set of groundwater problems that may contain a model of Long Island currently being developed at SUNY-Buffalo. In addition, there are many relevant optimization problems for circuit and device model codes at Sandia.

References

[1] D. P. Ahfeld, R. Page, and G. F. Pinder. Optimal ground-water remediation methods applied to a superfund site: from formulation to implementation. *Ground Water*, 33(1):58–70, 1995.

[2] D. P. Ahlfeld and M. Heidari. Applications of optimal hydraulic control to groundwater systems. *J. of Water Resources Planning and Management*, 120(3):350–365, 1994.

[3] D. P. Ahlfeld and M. P. Sprong. Presence of nonconvexity in groundwater concentration response functions. *J. of Water Resources Planning and Management*, 124(1):8–14, 1998.

[4] Mohamed H. Baker and John W. Bandler. An introduction to the space mapping technique. *Optimization and Engineering*, 2:369–384, 2001.

[5] John W. Bandler, Mostafa A. Ismail, Jose Ernesto Rayas-Sanchez, and Qi-Jun Zhang. Neuromodeling of microwave circuits exploiting space-mapping technology. *IEEE Transactions on Microwave Theory and Techniques*, 47(12):2417–2427, 1999.

[6] A. Battermann, J.M. Gablonsky, A. Patrick, C.T. Kelley, and K.R. Kavanagh. Solution of a groundwater flow problem with implicit filtering. *Optimization and Engineering*, 3:189–199, 2002.

[7] Kevin H. Brown, J. Richard Koteras, Don B. Longcope, and Thomas L. Warren. Cavity expansion: A library for cavity expansion algorithms, version 1.0. Technical Report SAND2003-1048, Sandia National Laboratories, Albuquerque, NM 87185, April 2003.

[8] J. E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM J. Opt.*, 1:448–474, 1991.

[9] E. D. Dolan, R. M. Lewis, and V. Torczon. On the local convergence properties of parallel pattern search. Technical Report 2000-36, NASA Langley Research Center, Institute for Computer Applications in Science and Engineering, Hampton, VA, 2000.

[10] Michael S. Eldred, Laura P. Swiler, David M. Gay, Shannon L. Brown, Anthony A. Giunta, William E. Hart, and Jean-Paul Watson. DAKOTA: A multilevel parallel

object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis - version 3.3 reference manual. Technical Report SAND2001-3515, Sandia National Laboratories, Albuquerque, NM 87185, December 2004.

[11] K.R. Fowler, C.T. Kelley, C.E. Kees, and C. T. Miller. A hydraulic capture application for optimal remediation design. In C.T. Miller, M.W. Farthing, W.G. Gray, and G.F. Pinder, editors, *Proceedings of the XV International Conference on Computational Methods in Water Resources*, 2004.

[12] K.R. Fowler, C.T. Kelley, C.T. Miller, M. S. Reed, C.E. Kees, and R. W. Darwin. Solution of a well field design problem with implicit filtering. to appear in Optimization and Engineering, 2003.

[13] David M. Gay. Usage summary for selected optimization routines. Technical Report Computing Science Technical Report No. 153, AT&T Bell Laboratories, Murray Hill, NJ 07974, October 1990.

[14] Anthony A. Giunta. Use of data sampling, surrogate models, and numerical optimization in engineering design. In *Proceedings of 40th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2002-0538, pages 1–13. AIAA, 2002.

[15] Genetha A. Gray and Tamara G. Kolda. APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization. Technical Report SAND2004-6391, Sandia National Laboratories, Livermore, CA 94551, August 2004.

[16] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comp.*, 22:789–828, 1996.

[17] W. D. Gropp and E. Lusk. User's guide for `mpich`, a portable implementation of MPI. Technical Report ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[18] P. D. Hough, T. G. Kolda, and V. Torczon. Asynchrounous parallel pattern search for nonlinear optimization. *SIAM J. Sci. Comput.*, 23:134–156, 2001.

[19] Tamara G. Kolda. Revisiting asynchronous parallel pattern search. Technical Report SAND2004-8055, Sandia National Laboratories, Livermore, CA 94551, February 2004.

[20] Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[21] Tamara G. Kolda and Virginia J. Torczon. Understanding asynchronous parallel pattern search. In Gianni Di Pillo and Almerico Murli, editors, *High Performance Algorithms and Software for Nonlinear Optimization*, volume 82 of *Applied Optimization*, pages 316–335. Kluwer Academic Publishers, Boston, 2003.

[22] Tamara G. Kolda and Virginia J. Torczon. On the convergence of asynchronous parallel pattern search. *SIAM Journal on Optimization*, 14(4):939–964, 2004.

[23] R. M. Lewis and V. Torczon. Rank ordering and positive bases in pattern search algorithms. Technical Report 96–71, Institute for Computer Applications in Science and Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, Virginia 23681–2199, 1996.

[24] R. M. Lewis and V. Torczon. Rank ordering and positive basis in pattern search algorithms. Technical Report 96-71, NASA Langley Research Center, Institute for Computer Applications in Science and Engineering, Hampton, VA, 1996.

[25] Robert Michael Lewis, Virginia Torczon, and Michael W. Tross et. Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, 124(1–2):191–207, December 2000.

[26] A.S. Meyer, C.T. Kelley, and C.T. Miller. Electronic supplement to "optimal design for problems involving flow and transport in saturated porous media". *Advances in Water Resources*, 12:1233–1256, 2002.

[27] A.S. Meyer, C.T. Kelley, and C.T. Miller. Optimal design for problems involving flow and transport in saturated porous media. *Advances in Water Resources*, 12:1233–1256, 2002.

[28] A. E. Mulligan and D. P. Ahlfeld. Advective control of groundwater contaminant plumes: Model development and comparison to hydraulic control. *Water Resources Research*, 35(8):2285–2294, 1999.

[29] T. D. Robinson, M. S. Eldred, K. E. Wilcox, and R. Haimes. Strategies for multifidelity optimization with variable dimensional hierarchical models. In *abstract submitted for 47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference (2nd AIAA Multidisciplinary Design Optimization Specialist Conference)*, Newport, Rhode Island, May 1-4, 2006.

[30] H. H. Rosenbrock. An automatic method for finding the greatest and least value of a function. *Computer Journal*, 3:175–184, 1960.

[31] Klaus Schittkowski. More test problems for nonlinear programming codes. *Lecture Notes in Economics and Mathematical Systems*, 282:118–223, 1987.

[32] Gregory D. Sjaardema. APREPRO: An algebraic preprocessor for parameterizing finite element analyses. Technical Report SAND92-2291, Sandia National Laboratories, Albuquerque, NM 87185, March 1997.

[33] L. M. Taylor and D. P. Flanagan. Pronto 3D a three-dimensional transient solid dynamics program. Technical Report SAND87-1912, Sandia National Laboratories, Albuquerque, NM 87185, December 1992.

[34] V. Torczon. PDS: Direct search methods for unconstrained optimization on either sequential or parallel machines. Technical Report TR92-09, Rice University, Department of Computational & Applied Math, Houston, TX, 1992.

[35] V. Torczon. On the convergence of pattern search algorithms. *SIAM J. Opt.*, 7:1–25, 1997.

[36] Thomas L. Warren and Mazen R. Tabbara. Spherical cavity-expansion forcing function in pronto 3d for application to penetration problems. Technical Report SAND97-1174, Sandia National Laboratories, Albuquerque, NM 87185, May 1997.

[37] Margaret H. Wright. Direct search methods: Once scorned, now respectable. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis)*, volume 344 of *Pitman Research Notes in Mathematics*, pages 191–208. CRC Press, 1996.

[38] C. Zheng, M. C. Hill, and P. A. Hsieh. *MODFLOW2000, The U.S.G.S Survey Modular Ground-Water Model User Guide to the LMT6 Package, the Linkage With MT3DMS for Multispecies Mass Transport Modeling*. USGS, user's guide edition, 2001.

[39] C. Zheng and P. P. Wang. *MT3DMS: A Modular Three-Dimensional Multispecies Transport Model for Simulation of Advection, Dispersion, and Chemical Reactions of Contaminants in Groundwater Systems*, documentation and user's guide edition, 1999.

Appendix A

Inner Loop Scripts

Here, we give explicit examples of scripts written to carry out our MFO scheme. These scripts are included to elucidate some of difficult details found in this document. All scripts were executed in a csh environment. Both PERL commands and Java calls to external parsers (for generating the aprepro type files) were used. The following scripts and files were included in the top level directory structure:

- high_fi_script – calculates the high fidelity response.
- low_fi_script – calculates the low fidelity response.
- interface_template_script – placeholder for many of the default advanced user options (changes made be expert users only).
- mfo_script – primary multifidelity optimization script.
- RUN_MFO_SCRIPT – high level run script.
- USER_INTERFACE_SCRIPT – placeholder for options that are to be modified by the user for each application.
- APPSPACK_INPUT_FILE – APPSPACK input file.
- response_template.txt – m4 file used to extract the relevant response values.
- **templatedir_highfi** – directory containing files required to complete a high fidelity function calculation.
  - high_fi_input_template.txt – file containing function input in aprepro format.
  - ParseFileToAprepro.class – java parser file that creates many of the aprepro files.
  - application specific auxiliary files required for a high fidelity calculation; includes simulators.

- **templatedir_lowfi** – directory containing files required to complete a low fidelity function calculation.
  - low_fi_input_template.txt – file containing function input in aprepro format.
  - ParseFileToAprepro.class
  - application specific auxiliary files including simulators.
- **templatedir_ls** – directory containing support files for the least squares calculation that is part of the space mapping routine.
  - dakota_least_square_template_dependent.in – DAKOTA run file.
  - least_squares_script_dependent – script file called directly by DAKOTA to do least squares calculation.
  - ParseFileToAprepro.class
- **templatedir_norm** – directory containing support files for the normalization routine.
  - dakota_param_template_1.in – script for completing a simple run of multiple low fidelity responses.
  - normalization_template_1.txt – script that calculates the normalization constant.
- **templatedir_opt** – directory for storing the support files necessary to do space mapped low fidelity optimization calculation.
  - apps_run_script_template – script that converts the APPSPACK input file to an aprepro style file.
  - appsinput_template.apps – template of the APPSPACK input file.
  - dakota_lowfi_opt_template.in – template of the DAKOTA run option.
  - ParseFileToAprepro.class

In the remainder of the Appendix, we include examples of the scripts mentioned above. The examples shown are customized for the groundwater remediation problem discussed in Section 5.2.

A.1 Main Execution Script

The top level run script is RUN_MFO_SCRIPT which is a user modifiable script. This script controls the entire MFO process by making the initial call to APPSPACK. Therefore, both the apps_input_file and the executable call must be set appropriately. Note that this script assumes that the user will execute the parallel version of APPSPACK.

BEGIN RUN_MFO_SCRIPT

```
#!/bin/csh -f
#################################################################
# the following arguments are called for from script in APPSPACK
# argv[1] = input file name
# argv[2] = output file name
# argv[3] = tag
# argv[4] = message (yes or no)
# argv[5] = fidelity
# argv[6] = number of variables
# argv[7..n] = tag name for each variable
#################################################################

#-----------------------------------------------------
# USER MODIFICATION SECTION
#-----------------------------------------------------

## set the name of the apps input file
set apps_input_file = "groundwater.apps"

# set the file that the apps output to screen will be piped into
set apps_output_file = "groundwater.out"

# set the number of processors
@ number_processors = 4

# set the apps executable (include path if necessary)
set apps_exe = "/home/gagray/appspack/src/appspack_mpi"

#################################################################
#################################################################
##############DO NOT MAKE ANY CHANGES BELOW HERE
#################################################################
#################################################################

#
# do some checking of the apps input file
#

# check that cache output file is set to apps_cache
fgrep -w "Cache Output File" $apps_input_file >! tmp.apps
fgrep -w "apps_cache" tmp.apps >! tmp2.apps
set num_file_lines = ‘less tmp2.apps | wc -l‘
```

```
if ($num_file_lines == 0) then
  echo "*****  RUN_MFO_SCRIPT ERROR *****"
  echo \"Cache Output File\" "must be set to" \"apps_cache\" \\
   "for the MFO routine to work correctly.  \\
    Please add or reset this option."
  exit
endif

#check that correct interface script is used
fgrep -w "Executable Name" $apps_input_file >! tmp.apps
fgrep -w "USER_SCRIPT" tmp.apps >! tmp2.apps
set num_file_lines = `less tmp2.apps | wc -l`

if ($num_file_lines == 0) then
  echo "*****  RUN_MFO_SCRIPT ERROR *****"
  echo \"Executable Name\" "must be set to" \"USER_SCRIPT\" \\
   "for the MFO routine to work correctly.  \\
    Please add or reset this option."
  exit
endif

# set up the USER_SCRIPT script for the given apps input file
rm -f USER_SCRIPT
m4 -DAPPS_INPUT_FILE="$apps_input_file" \\
   -DAPPS_OUTPUT_FILE="$apps_output_file" \\
   USER_INTERFACE_SCRIPT >! USER_SCRIPT
chmod a+x USER_SCRIPT

#
# do some clean up
#
if ( -e number_calcs) rm number_calcs
rm tmp.apps tmp2.apps
rm -f interface_script_*

#
# begin execution of appspack in background
#
mpirun -np $number_processors $apps_exe $apps_input_file \\
  >! $apps_output_file &
```

END RUN_MFO_SCRIPT

A.2 User Interface Scripts

This section contains the scripts that require application specific modifications by the user. The low_fi_script and high_fi_script only require changes for a certain applications. The USER_INTERFACE_SCRIPT is the primary file for user options of the MFO scheme.

A.2.1 User Input Scripts

The user interface script, USER_INTERFACE_SCRIPT, controls the run options of the MFO calculations. Most of the user modifications are done in this top level script file. Some advanced user options are defined in interface_script (not shown here). A cleaner implementation should be done in the future so that all user options are defined in a single input file.

BEGIN USER_INTERFACE_SCRIPT

```
#!/bin/csh -f
#
#
# This script is the inner loop of the MFO algorithm w/Space Mapping
# and is called by an APPSPACK worker node (NOTE: this may be
# generalized as an inner loop for GA as well)
#
# The following arguments are passed directly from APPSPACK
# argv[1] = function input file name
# argv[2] = function output file name
# argv[3] = function input tag number
#
#######################################################
#######################################################
#######################################################
###### PRE-PROCESSING SECTION DO NOT CHANGE
#######################################################
#######################################################
# macros set up for the m4 call
# changequote(,) removes the dependence on ‘ as a quote character
changequote(,)
set apps_input_file = "APPS_INPUT_FILE"
set apps_output_file = "APPS_OUTPUT_FILE"

set apps_input = $argv[1]
set apps_output = $argv[2]
set apps_tag = $argv[3]
```

```
#########################################################
#########################################################
######USER MODIFICATIONS REQUIRED IN THIS SECTION
#########################################################
#########################################################
### SET the number of responses and the minimum number of
# calculations between each inner loop call (min_calcs)
@ num_responses = 2
@ min_calcs = 2

### SET the high fidelity variable tags
# high_var_tag_option = <sequential|explicit>
#  - sequential - the single tag name, defined in high_var_tag,
#                 will have a sequential number (starting at 0)
#                 appended to the end of the tag
#  - explicit - all tags defined by the user in high_var_tag
#
# high_var_tag - if sequential, place a single tag value
#                if explicit, place the tags in arrays
#                (tag_1 tag_2 tag_3 ...)

set high_var_tag_option = sequential
set high_var_tag = x

### SET the number of variables, variable tags, and value ranges
# for the low fidelity models
#
# low_var_tag_option = <one_to_one|sequential|explicit>
#  - one_to_one - one to one mapping from high to low so
#                 parameters are defined equivalently.
#  - sequential - the single tag name defined in low_var_tag will
#                 have a sequential number (starting at 0)
#                 appended to the end of the tag.
#  - explicit - all tags defined by user in low_var_tag
#
# num_low_var = number of low fidelity model variables; Note that
#               this need not be equivalent to the number of high
#               fidelity variables & thus must be defined except
#               for the one_to_one case
#
# low_var_tag = if sequential defined, place a single tag value
#               if explicit defined, place the tags in arrays
#               (tag_1 tag_2 tag_3 ...).
#               not needed if one_to_one is set.
```

```
set low_var_tag_option = one_to_one
#set num_low_var = 12
#set low_var_tag = x

### SET the space mapping parameter tags and values used for fitting
### SET the range and init values of the fitting parameters
#
# Choose one of the following three
# (depends on the number of space map vars)
#
# space_map_tag_option = <global|explicit>
#  - global - the low/high/init values are equivalently defined for
#             a given space_map_tag for all variables
#  - explicit - all tags must be defined by user in space_map_tag
#               and all associated low/high/inti values. Note: each
#               design variable is assigned to each space param tag.
#
# num_space_map = the number of space parameter; if global,
#                 no need to set this parameter
#
# space_map_tag = If global, define each space_map_tag without
#                  the high_var_tag appendix  Ex.  (alpha, beta)
#                 If explicit, every space_map_tag in the
#                  high_var_tag appendix must be defined.
#                  Ex. (alpha_x0 alpha_x1 beta_x0 beta_x1)
#
# space_map_low = lower bounds of space_map_tag.
#                 If global, define each value in the order of
#                  space_map_tag  Ex. (alpha, beta) -> (0.0 1.0)
#                 If explicit, define each value in the order of
#                  space_map_tag
#                  Ex. (alpha_x0 alpha_x1 beta_x0 beta_x1) ->
#                       (0.0  0.0  1.0  1.0)
#
# space_map_high = higher bounds of the space_map_tag. See
#                    space_map_low for specifications.
#
# space_map_init = initial value of the space_map_tag. See
#                    space_map_low for specifications.
#
######### alpha
set space_map_tag_option = global
set space_map_tag = (alpha)
set space_map_high = (10)
```

```
set space_map_low = (0.5)
set space_map_init = (1.0)

######### alpha and gamma
#set space_map_tag_option = global
#set space_map_tag = (alpha gamma)
#set space_map_low =  (0.5 0.5)
#set space_map_high = (10.0 10.0)
#set space_map_init = (1.0 1.0)

######### alpha, gamma, and beta
#set space_map_tag_option = global
#set space_map_tag = (alpha gamma beta)
#set space_map_low =  (0.5 0.5 1.0)
#set space_map_high = (10.0 10.0 2.0)
#set space_map_init = (1.0 1.0 1.0)

### SET the low-fidelity optimization strategy
#set opt_strategy = grad
set opt_strategy = apps

#########################################################
#########################################################
###############DO NOT MAKE ANY CHANGES BELOW HERE
#########################################################
#########################################################
# get the number of high fidelity variables and the
# initial best point from the apps function input file
# which has the following format:
#
#  <number_variables>
#  <variable_1>
#      .
#      .
#  <variable_n>
#  <yes|no>

set value = ‘less $apps_input‘
set num_high_var = $value[1]

# create a string with all high fi vars (high_var_tag_string)
switch ($high_var_tag_option)
case sequential:
    @ ii = 0
    set high_var_tag_string = " "
```

```
    while ($ii < $num_high_var)
set high_var_tag_index = \\
          'perl -e "{ print $high_var_tag . $ii }"'
set high_var_tag_string = 'echo "$high_var_tag_string \\
  $high_var_tag_index "'
@ ii++
    end
    breaksw
case explicit:
    if ($num_high_var != $#high_var_tag) then
      echo "***** USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of high_var_tag should equal \\
       $num_high_var but is currently equal to \\
       $#high_var_tag. Please edit this parameter."
      exit
    endif
    set high_var_tag_string = 'echo "  $high_var_tag  "'
    breaksw
default:
    echo ***** USER_INTERFACE_SCRIPT ERROR *****"
    echo "An improper high_var_tag_option has been set to \\
    $high_var_tag_option.  Please change to sequential or explicit."
    exit
    breaksw
endsw

# create a string with all the low fi variables (low_var_tag_string)
switch ($low_var_tag_option)
case one_to_one:
    set num_low_var = $num_high_var
    set low_var_tag_string = 'echo "$high_var_tag_string"'
    breaksw
case sequential:
    if (!($?num_low_var)) then
        echo ***** USER_INTERFACE_SCRIPT ERROR *****"
echo "The num_low_var must be set with the sequential \\
          option, please set this variable"
exit
    endif
    if (!($?low_var_tag)) then
        echo *****  USER_INTERFACE_SCRIPT ERROR *****"
echo "The low_var_tag must be set with the sequential option, \\
please set this variable"
exit
    endif
```

```
    @ ii = 0
    set low_var_tag_string = " "
    while ($ii < $num_low_var)
set low_var_tag_index = \\
          'perl -e "{ print $low_var_tag . $ii }"'
set low_var_tag_string = 'echo "$low_var_tag_string \\
  $low_var_tag_index "'
@ ii++
    end
    breaksw
case explicit:
    if (!($?num_low_var)) then
        echo ***** USER_INTERFACE_SCRIPT ERROR *****"
echo "the num_low_var must be set with the explicit \\
  option, please set this variable"
exit
    endif
    if (!($?low_var_tag)) then
        echo *****  USER_INTERFACE_SCRIPT ERROR *****"
echo "the low_var_tag must be set with the explicit \\
  option, please set this variable"
exit
    endif
    if ($num_low_var != $#low_var_tag) then
      echo *****  USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of low_var_tag should equal $num_low_var \\
       but is currently equal to $#low_var_tag. \\
       Please edit this parameter."
      exit
    endif
    set low_var_tag_string = 'echo "  $low_var_tag  "'
    breaksw
default:
    echo ***** USER_INTERFACE_SCRIPT ERROR *****"
    echo "An improper low_var_tag_option has been set to \\
      $low_var_tag_option. Please change to one_to_one, \\
      sequential or explicit."
    exit
    breaksw
endsw

# create a list of space parameter strings
set space_map_tag_string = " "
set space_map_low_string = " "
set space_map_high_string = " "
```

```
set space_map_init_string = " "

# create a string with all space map vars (space_map_tag_string)
switch ($space_map_tag_option)
case global:
    # set the number of space parameters
    set num_space_map_tag = $#space_map_tag
    @ num_space_map = $num_high_var * $num_space_map_tag

    # make sure the number of values is equivalent to the number
    # of parameters
    if ($num_space_map_tag != $#space_map_high) then
      echo *****  USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of space_map_high parameters should equal \\
        $num_space_map_tag but is currently equal to \\
        $#space_map_high. Please edit this parameter."
      exit
    endif

    if ($num_space_map_tag != $#space_map_low) then
      echo *****  USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of space_map_low parameters should equal \\
        $num_space_map_tag but is currently equal to \\
        $#space_map_low. Please edit this parameter."
      exit
    endif

    if ($num_space_map_tag != $#space_map_init) then
      echo ***** USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of space_map_init parameters should equal \\
        $num_space_map_tag but is currently equal to \\
        $#space_map_init. Please edit this parameter."
      exit
    endif

    # set the space_map_strings
    @ ii = 0
    while ($ii < $num_space_map_tag)
        @ jj = 0
while ($jj < $num_high_var)
    @ iindex = $ii + 1
    @ jindex = $jj + 1
    set space_map_tag_index =  \\
    ‘perl -e "{ print $space_map_tag[$iindex] . _ .  \\
    $high_var_tag_string[$jindex] }"‘
```

```
    set space_map_tag_string = \\
    ‘echo "$space_map_tag_string $space_map_tag_index "‘
    set space_map_high_string = \\
    ‘echo "$space_map_high_string $space_map_high[$iindex] "‘
    set space_map_low_string = \\
    ‘echo "$space_map_low_string $space_map_low[$iindex] "‘
    set space_map_init_string =  \\
    ‘echo "$space_map_init_string $space_map_init[$iindex] "‘
    @ jj++
end
@ ii++
    end

    breaksw
case explicit:

    #error checks
    if (!($?num_space_map)) then
      echo ***** USER_INTERFACE_SCRIPT ERROR *****"
      echo "the parameter num_space_map must be set when explicit \\
        is set as space_map_tag_option. Please edit this parameter."
      exit
    endif

    if ($num_space_map != $#space_map_tag) then
      echo ***** USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of space_map_tag parameters should equal \\
        $num_space_map but is currently equal to \\
        $#space_map_tag. Please edit this parameter."
      exit
    endif

    if ($num_space_map != $#space_map_high) then
      echo ***** USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of space_map_high parameters should equal \\
        $num_space_map but is currently equal to \\
        $#space_map_high.  Please edit this parameter."
      exit
    endif

    if ($num_space_map != $#space_map_low) then
      echo ***** USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of space_map_low parameters should equal \\
        $num_space_map but is currently equal to \\
        $#space_map_low.  Please edit this parameter."
```

```
      exit
    endif

    if ($num_space_map != $#space_map_init) then
      echo ***** USER_INTERFACE_SCRIPT ERROR *****"
      echo "the number of space_map_init parameters should equal \\
        $num_space_map but is currently equal to \\
        $#space_map_init. Please edit this parameter."
      exit
    endif

    # set the variables
    set space_map_tag_string = ‘echo "  $space_map_tag  "‘
    set space_map_high_string = ‘echo "  $space_map_high  "‘
    set space_map_low_string = ‘echo "  $space_map_low  "‘
    set space_map_init_string = ‘echo "  $space_map_init  "‘

    breaksw
default:
    echo ***** USER_INTERFACE_SCRIPT ERROR *****"
    echo "An improper space_map_tag_option has been set to \\
    $space_map_tag_option.  Please change to global or explicit."
    exit
    breaksw
endsw

# get the low_fi bounds and best point
# NOTE: should be moved into a separate script in the future

# set the init value to the current best point
set low_var_init_string = " "
set low_var_high_string = " "
set low_var_low_string = " "
set low_value_low = ‘fgrep -w "Lower" $apps_input_file‘
set low_value_high  = ‘fgrep -w "Upper" $apps_input_file‘

# error checks
@ num_low_var_init_current = $#value - 2
@ num_low_var_high_current = $#low_value_low - 3
@ num_low_var_low_current = $#low_value_high - 3

if ($num_low_var != $num_low_var_init_current) then
    echo *****  USER_INTERFACE_SCRIPT ERROR *****"
    echo "the number of low_var_init parameters should equal \\
      $num_low_var but is currently equal to \\
```

```
      $num_low_var_init_current. \\
      There may be an error within $apps_input."
    exit
endif

if ($num_low_var != $num_low_var_high_current) then
    echo ***** USER_INTERFACE_SCRIPT ERROR *****"
    echo "the number of low_var_high parameters should equal \\
      $num_low_var but is currently equal to \\
      $num_low_var_high_current. \\
      There may be an error within $apps_input_file."
    exit
endif

if ($num_low_var != $num_low_var_low_current) then
    echo ***** USER_INTERFACE_SCRIPT ERROR *****"
    echo "the number of low_var_low parameters should equal \\
      $num_low_var but is currently equal to \\
      $num_low_var_low_current. \\
      There may be an error within $apps_input_file."
    exit
endif
# set the values
@ ii = 0
while ($ii < $num_high_var)
    @ iindex = $ii + 2
    @ jindex = $ii + 4
    set low_var_init_string = \\
    ‘echo "$low_var_init_string $value[$iindex] "‘
    set low_var_high_string = \\
    ‘echo "$low_var_high_string $low_value_high[$jindex] "‘
    set low_var_low_string = \\
    ‘echo "$low_var_low_string $low_value_low[$jindex] "‘
    @ ii++
end

#create interface_script_tag
m4 -DHIGH_VAR_TAG_STRING="$high_var_tag_string" \\
-DLOW_VAR_TAG_STRING="$low_var_tag_string" \\
-DSPACE_MAP_TAG_STRING="$space_map_tag_string" \\
-DSPACE_MAP_HIGH_STRING="$space_map_high_string" \\
-DSPACE_MAP_LOW_STRING="$space_map_low_string" \\
-DSPACE_MAP_INIT_STRING="$space_map_init_string" \\
-DNUM_HIGH_VAR="$num_high_var" \\
-DNUM_LOW_VAR="$num_low_var" \\
```

```
-DNUM_SPACE_MAP="$num_space_map" \\
-DLOW_VAR_HIGH_STRING="$low_var_high_string" \\
-DLOW_VAR_LOW_STRING="$low_var_low_string" \\
-DLOW_VAR_INIT_STRING="$low_var_init_string" \\
interface_template_script >! interface_script_$apps_tag

chmod a+x interface_script_$apps_tag

# call interface_script_tag
interface_script_$apps_tag $apps_input $apps_output \\
$apps_tag $num_responses $min_calcs $opt_strategy

# remove the interface script when finished
rm interface_script_$apps_tag
```

END USER_INTERFACE_SCRIPT

A.2.2 Low Fidelity Model Scripts

This section includes the low fidelity script file for the groundwater remediation problem. Note that there are only two lines of the script that require user modification. Therefore, future work would include moving these modifications to the top level user scripts.

BEGIN low_fi_script

```
#!/bin/csh -f
#
#############################################################
# $argv[1] is params.in.(fn_eval_num) from DAKOTA
# $argv[2] is results.out.(fn_eval_num) returned to DAKOTA
#############################################################

#-----------------------------------------------
# PRE-PROCESSING : NO USER MODIFICATIONS REQUIRED
#-----------------------------------------------

set num = `echo $argv[1] | cut -c 11-`

cp -r templatedir workdir.$num
mv $argv[1] workdir.$num/dakota_vars
cd workdir.$num
```

```
# run a different executable if least squares option is passed
# into this script (argv[3] == "ls")
set executable =  APPSfbhc
if (($#argv > 2) && ($argv[3] == "ls")) \\
  set executable = APPSfbhc_ls

# --------
# ANALYSIS AND OUTPUT
# --------

##################################################
# --------- USER MODIFICATION REQUIRED -----------
##################################################
# Change the file templatedir_lowfi/low_fi_input_template.txt
# so that low_fi_input.in is in the appropriate format for
# your application; No further applications are needed
aprepro -qW low_fi_input_template.txt temp_response
perl -ne '{print unless /^\s*$/o}' temp_response > low_fi_input.in
rm temp_response

##################################################
# --------- USER MODIFICATION REQUIRED -----------
##################################################
# add the appropriate low fidelity executable call below
./$executable low_fi_input.in response.txt $num

##################################################
# NO USER MODIFICATIONS REQUIRED BELOW HERE
##################################################

# places response.txt into dakota file
cat response.txt > $argv[2]

# NOTE: moving $argv[2] at the end of the script avoids any
# problems with read race conditions
# (although master-slave does not have this problem).
mv $argv[2] ../.
cd ..

# -------------
# CLEANUP
# -------------
rm -rf workdir.$num
```

END low_fi_script

A.2.3 High Fidelity Model Scripts

Below is the high fidelity script file for the groundwater remediation problem. As with the low fidelity script, there are only two possible lines for user modification, and the plan is to move this to the top level user scripts in the future.

BEGIN hi_fi_script

```
#!/bin/csh -f
#################################################################
# the following arguments are called for this script in APPSPACK
# argv[1] = input file name
# argv[2] = output file name
# argv[3] = tag
# argv[4] = message (yes or no)
# argv[5] = fidelity
# argv[6] = number of variables
# argv[7..n] = tag name for each variable
#################################################################

#-----------------------------------------------------
# PRE-PROCESSING
# NO USER MODIFICATIONS REQUIRED IN THIS SECTION
#-----------------------------------------------------

## just set num to the tag value
set num = $argv[3]
set message = $argv[4]
set fidelity = $argv[5]
set number_var = $argv[6]
@ start_index = 7

# be sure to set unique directories
while ( -e workdir_$fidelity.$num)
    set num = $num.$argv[3]
end

# cp the templatedir and needed files into
cp -r templatedir_$fidelity workdir_$fidelity.$num
mv $argv[1] workdir_$fidelity.$num
cp response_template.txt workdir_$fidelity.$num
cd workdir_$fidelity.$num

# create a string list of variables
```

```
@ iv = 0
set var_string = " "
while ($iv < $number_var)

    @ var_index = $iv + $start_index
    set var_string = ‘echo "$var_string $argv[$var_index] "‘
    @ iv++
end

## process the appspack input file into a aprepro file
java ParseFileToAprepro input $argv[1] $number_var \\
  $var_string dakota_vars

# -------------------
# ANALYSIS AND OUTPUT
# -------------------

#########################################################
# --------- USER MODIFICATION REQUIRED -----------
#########################################################
# NOTE: Modify templatedir_highfi/high_fi_input_template.txt
# so that high_fi_input.in has the appropriate format for your
# application; No other modification required below
aprepro -qW high_fi_input_template.txt temp_response
perl -ne ’{print unless /^\s*$/o}’ temp_response > high_fi_input.in
rm temp_response

#########################################################
# --------- USER MODIFICATION REQUIRED -----------
#########################################################
# add the appropriate high fidelity executable call below
/home/gagray/groundwater/apps-call/APPSComHC high_fi_input.in \\
response.txt $argv[3]

#########################################################
#NO USER MODIFICATIONS REQUIRED BELOW
#########################################################
#cat all require values into the dakota output file
echo $message > message.txt
cat response.txt message.txt  > $argv[2]
cp $argv[2] ../.


switch ($message)
case yes:
```

```
    # place each parameter value within file
    @ ii = 0
    while ($ii < $number_var)

@ varindex = $ii + $start_index

m4 -DVAR_FILE="dakota_vars" -DVAR_NAME=$argv[$varindex] \\
  response_template.txt > temp_file.txt
aprepro temp_file.txt temp.new
grep -vi aprepro temp.new > temp_.new

# removes Aprepro line and first blank line
perl -ne \\
         '{print unless /^\s*$/o}' temp_.new | cat >> point.$argv[3]


@ ii++
    end

    # make point file writable by everyone in case job hangs & point
    # file is left in tmp directory
    chmod a+w point.$argv[3]
    mv point.$argv[3] /tmp/point.$argv[3]
    breaksw
case no:
    breaksw
endsw

# move back out of the work directory
cd ..

# -------------
# CLEANUP
# -------------

#rm -rf workdir_$fidelity.$num
```

END hi_fi_script

A.3 Multifidelity Optimization Script

This script is the primary control script of the MFO calculation, and it directly calls most of the other scripts. If additional optimization schemes or mapping approaches are developed in the future, this script will encapsulate these changes.

BEGIN mfo_script

```
#!/bin/csh -f
#
#########################################################################
# This script is the inner loop of the MFO algorithm w/Space Mapping
# and is called by an APPSPACK worker (NOTE: this may be generalized
# as an inner loop for GA as well)
#
# the arguments are
# argv[1] = input file name
# argv[2] = output file name
# argv[3] = tag
# argv[4] = message (yes or no)
# argv[5] = fidelity
# argv[6] = response tag
# argv[7] = number of low fidelity variables
# argv[8..n1] = tag name for each low fidelity variable
# argv[n1+1] = number of high fidelity variables
# argv[(n1+1)..n2] = tag name for each high fidelity variable
#########################################################################
##### NO USER MODIFICATIONS IN THIS SCRIPT ##########################
#########################################################################

# make an inner loop directory an place everything within here
## get all argv values a name in case additional values are added
set num = $argv[3]
set message = $argv[4]
set fidelity = $argv[5]
set normalization = $argv[6]
set response_tag = $argv[7]
set number_responses = $argv[8]
set opt_strategy = $argv[9]
set ls_calc_type = $argv[10]
set num_high_var = $argv[11]
@ start_high_index = 12
@ low_index_var = $start_high_index + $num_high_var
@ start_low_index = $low_index_var + 1
```

```
set num_low_var = $argv[$low_index_var]
@ space_param_index_var = $start_low_index + ($num_low_var * 4)
@ start_space_param_index = $space_param_index_var + 1
set num_space_param = $argv[$space_param_index_var]

while ( -e inner_loop.$num)
    set num = $num.$argv[3]
end

# move template, cache, and script files into inner loop
mkdir innerloop.$num
cp response_template.txt innerloop.$num
cp apps_cache innerloop.$num
cp low_fi_script innerloop.$num

cd innerloop.$num

#---------------
# Get APPSPACK data
#---------------

# get the apps data (the high fidelity model runs) and make aprepro
# friendly file named apps_out_vars
cp -f ../templatedir_$fidelity/ParseFileToAprepro.class .

# create a string with all high fi variables (high_var_tag_string)
@ iv = 0
set high_var_tag_string = " "
while ($iv < $num_high_var)

    @ var_index = $iv + $start_high_index
    set high_var_tag_string = 'echo "$high_var_tag_string \\
      $argv[$var_index] "'
    @ iv++
end

# create a string with all the low fi variables (low_var_tag_string)
@ iv = 0
set low_var_tag_string = " "
set low_var_tag_wparen_string = " "
set low_var_low_string = " "
set low_var_high_string = " "
set low_var_init_string = " "
while ($iv < $num_low_var)
```

```
    @ varindex = $iv + $start_low_index
    set low_var_tag_string = ‘echo "$low_var_tag_string \\
      $argv[$varindex] "‘
    set low_var_tag_wparen_string = \\
      ‘echo "$low_var_tag_wparen_string ’$argv[$varindex]’ "‘
    @ varindex += $num_low_var
    set low_var_low_string = ‘echo "$low_var_low_string \\
      $argv[$varindex] "‘
    @ varindex += $num_low_var
    set low_var_high_string = ‘echo "$low_var_high_string \\
      $argv[$varindex] "‘
    @ varindex += $num_low_var
    set low_var_init_string = ‘echo "$low_var_init_string \\
      $argv[$varindex] "‘

    @ iv++
end

# create list of var string values to be used in script
@ m=0
set hifi_design_value_string = " "
while ($m < $number_responses)

    @ mindex = $m + 1
    @ j=0

    while ($j < $num_high_var)
@ varindex = $j + $start_high_index
# set design_var to current design variable
set design_var = $argv[$varindex]
# concatenate string values into design variable index values
set designVarIndex = ‘perl -e \\
          "{ print $design_var . _ . $mindex }"‘
# create design variable value string
set hifi_design_value_string = \\
  ‘echo "$hifi_design_value_string { $designVarIndex } "‘
@ j++
    end
    @ m++
end

# create a list of space mapping var string values
set space_map_tag_string = " "
set space_map_low_string = " "
set space_map_high_string = " "
```

```
set space_map_init_string = " "

@ j=0
while ($j < $num_space_param)
    @ varindex = $j + $start_space_param_index
    # set space mapping strings
    set space_map_tag_string=‘echo "$space_map_tag_string \\
      $argv[$varindex]’ "‘
    @ varindex += $num_space_param
    set space_map_low_string=‘echo "$space_map_low_string \\
      $argv[$varindex] "‘
    @ varindex += $num_space_param
    set space_map_high_string=‘echo "$space_map_high_string \\
      $argv[$varindex] "‘
    @ varindex += $num_space_param
    set space_map_init_string=‘echo "$space_map_init_string \\
      $argv[$varindex] "‘
    @ j++
end

# debug output
##echo "high fidelity tags        = $high_var_tag_string"
##echo "low fidelity tags         = $low_var_tag_string"
##echo "low fidelity low values   = $low_var_low_string"
##echo "low fidelity high values  = $low_var_high_string"
##echo "low fidelity init values  = $low_var_init_string"
##echo "space mapping tags        = $space_map_tag_string"
##echo "space mapping low values  = $space_map_low_string"
##echo "space mapping high values = $space_map_high_string"
##echo "space mapping init values = $space_map_init_string"

# NOTE: THIS SHOULD BE CHANGED TO cache_message IN THE FUTURE AND
# THE PARSER SHOULD LOOK FOR NUMBER TAG
java ParseFileToAprepro cache apps_cache $num_high_var \\
$high_var_tag_string  apps_out_vars $response_tag $number_responses

#---------------
# Run Normalization Calculation (between high and low fidelity model)
#---------------

switch($normalization)
case on:
    # move all appropriate files to a new workdir_norm_$jindex
    # now a single normalization directory
    cp -r ../templatedir_norm workdir_norm
```

```
    cp apps_out_vars workdir_norm
    cp low_fi_script workdir_norm
    cp ../templatedir_lowfi/* workdir_norm
    cp -r ../templatedir_lowfi workdir_norm/templatedir
    cp apps_out_vars workdir_norm/templatedir
    cd workdir_norm

    # calculate normalization constant and tag unto apps_out_vars

    # get the list of var string names
    @ iv = 0
    set dakota_var_string = " "
    while ($iv < $num_high_var)

@ var_index = $iv + $start_high_index
set dakota_var_string = 'echo "$dakota_var_string '\\
          $argv[$var_index]' "'
@ iv++
    end

    # add DESIGN_VALUES and DESIGN_VARS terms
    m4 -DDESIGN_VARS="$dakota_var_string" \\
      -DDESIGN_VALUES="$hifi_design_value_string" \\
      dakota_param_template_1.in > dakota_param_template.in

    # cleanup
    rm -f dakota_param_template_*.in

    # substitute values from aprepro file
    aprepro -qW dakota_param_template.in dakota_param.in

    # do parameter study
    dakota -input dakota_param.in >>&! dakota_param.out

    # place response values in aprepro format (normalization_vars)
    java ParseFileToAprepro dakota_tabular_table \\
      dakota_tabular.dat 0 normalization_vars


    # loop over and average sum of ratio of low & high fidelity
    # response

    @ n=0
    while ($n < $number_responses)
```

```
@ nindex = $n + 1
@ nindexp1 = $nindex + 1

# insert the design variable
m4 -DRATIO_$nindex="\\
          (response_$nindex / response_fn1_$nindex) \\
  + RATIO_$nindexp1" normalization_template_$nindex.txt \\
  > normalization_template_$nindexp1.txt

@ n++
    end

    # clear out RATIO tags from the last insertion
    m4 -DRATIO_$nindexp1="0.0"  \\
      normalization_template_$nindexp1.txt \\
      > normalization_template.txt

    # substitute values from aprepro file
    aprepro -qW normalization_template.txt normal_out_vars

    # cleanup
    rm -f normalization_template_*.txt
    # add normalization variable to apps_out_vars and cp in
    # directory above
    cat normal_out_vars apps_out_vars > apps_out_vars_modified
    cp apps_out_vars_modified ../apps_out_vars

    cd ..
breaksw # case(on)
case off:

    # set the normalization_factor to 1.0
    echo "{ normalization_factor = 1.0 }" > normal_out_vars

    # add normalization variable to apps_out_vars
    cat normal_out_vars apps_out_vars > apps_out_vars_modified
    cp apps_out_vars_modified apps_out_vars

breaksw # case (off)
endsw # switch($normalization)

#---------------
# Run Least Squares Calculation
#---------------
```

```
switch($ls_calc_type)
case independent:
    #
    # BEGIN LOOP HERE FOR MULTIPLE DESIGN VARIABLES
    #

    @ j=0
    while ($j < $num_low_var)

@ jindex = $j + 1
@ jindexp1 = $jindex + 1
@ varindex = $j + $start_low_index

# set design_var to current design variable
set design_var = $argv[$varindex]

    #---------------
    # Run Least Squares Calculation
    #---------------

# move all least squares files to a new workdir_ls_$jindex
cp -r ../templatedir_ls workdir_ls_$jindex
cp apps_out_vars workdir_ls_$jindex
cp response_template.txt workdir_ls_$jindex
cp -r ../templatedir_lowfi workdir_ls_$jindex/templatedir
cp apps_out_vars workdir_ls_$jindex/templatedir
cd workdir_ls_$jindex


# convert dakota_least_squares_template.in to
        # dakota_least_squares.in; insert appropriate design
        # variable and response values of high
# fidelity model to do least squares calculation

@ i=0

while ($i < $number_responses)

    @ iindex = $i + 1
    @ iindexp1 = $iindex + 1

    # concatenate string values into design variable
            # index values
    set designVarIndex = `perl -e \\
              "{ print $design_var . _ . $iindex }"`
```

```
    # insert the design variable values and descriptors
    m4 -DCSV_VALUES_$iindex="{$designVarIndex} \\
      CSV_VALUES_$iindexp1" \\
      -DCSV_DESCRIPTORS_$iindex="'$designVarIndex' \\
      CSV_DESCRIPTORS_$iindexp1" \\
      dakota_least_squares_template_$iindex.in \\
      > dakota_least_squares_template_$iindexp1.in

    @ i++
end

# clear out CSV tags from last insertion
m4 -DCSV_VALUES_$iindexp1 -DCSV_DESCRIPTORS_$iindexp1 \\
dakota_least_squares_template_$iindexp1.in > \\
        dakota_least_squares_template_temp.in

# add CSV_TAGE
m4  -DCSV_TAG="design_variable_$jindex" \\
dakota_least_squares_template_temp.in > \\
        dakota_least_squares_template.in

# cleanup
rm -f dakota_least_squares_template_*.in

# substitute values from aprepro file
aprepro -qW dakota_least_squares_template.in \\
          dakota_least_squares.in

# do least squares calculation
dakota -input dakota_least_squares.in \\
          >>&! dakota_least_squares.out

# place alpha, beta, gamma within aprepro friendly format
# (space_mapping_vars)
java ParseFileToAprepro dakota_tabular \\
          dakota_tabular.dat 0 space_mapping_vars_$jindex

# move spacing mapping aprepro file up before cleaning dir
cp space_mapping_vars_$jindex ../.
cd ..

@ j++
    end
```

```
    # remove old space_mapping_vars
    rm space_mapping_vars

    # cat all the space_mapping_vars files into one
    @ jj=0
    while ($jj < $num_low_var)

    # parameter used for indexing
@ jjindex = 1 + $jj
cat space_mapping_vars_$jjindex >> space_mapping_vars
@ jj++
    end # while ($j < $num_low_var)
    breaksw #case independent
case dependent:
    #---------------
    # Run Least Squares Calculation
    #---------------

    # move all least squares files to a new workdir_ls_
    cp -r ../templatedir_ls workdir_ls
    cp apps_out_vars workdir_ls
    cp response_template.txt workdir_ls
    cp low_fi_script workdir_ls
    cp -r ../templatedir_lowfi workdir_ls/templatedir
    cp apps_out_vars workdir_ls/templatedir

    cd workdir_ls

    # convert dakota_least_squares_template.in to
    # dakota_least_squares.in; insert appropriate design vars &
    # response vals of high-fi model to do LS calculation
    set space_var_init = " "
    set space_var_upper = " "
    set space_var_lower = " "
    set space_var_descriptor = " "
    set design_var_descript = " "

    @ i=0
    while ($i < $number_responses)

@ iindex = $i + 1

@ j=0
while ($j < $num_low_var)
```

```
    @ varindex = $j + $start_low_index

        # concatenate string vals into design var index vals
        # set designVarIndex = `perl -e \\
        #   "{ print $design_var . _ . $iindex }"`
    set design_var_descript =  `echo \\
         "$design_var_descript '$argv[$varindex]_$iindex' "`

    @ j++
end

@ i++
    end

    set space_var_init        = `echo "$space_map_init_string"`
    set space_var_lower       = `echo "$space_map_low_string"`
    set space_var_upper       = `echo "$space_map_high_string"`
    set space_var_descriptor  = `echo "$space_map_tag_string"`
    set total_num_space_param = `echo "$num_space_param"`

# insert the design variable values and descriptors
    m4 -DCDV_INIT="$space_var_init" -DCDV_LOWER="$space_var_lower" \\
    -DCDV_UPPER="$space_var_upper" \\
    -DCDV_DESCRIPTOR="$space_var_descriptor" \\
    -DDESIGN_VARS="$low_var_tag_string" \\
    -DNUM_SPACE_PARAMS="$total_num_space_param"\\
    dakota_least_squares_template_dependent.in \\
    > dakota_least_squares_template.in

    # substitute values from aprepro file
    aprepro -qW dakota_least_squares_template.in \\
      dakota_least_squares.in

    # sourcing .cshrc file, may remove this in future
    source ~/.cshrc

    # do least squares calculation
    dakota -input dakota_least_squares.in >>&! \\
      dakota_least_squares.out

    # place alpha, beta, gamma within aprepro friendly format
    # (space_mapping_vars)
    java ParseFileToAprepro dakota_tabular dakota_tabular.dat 0 \\
    space_mapping_vars
```

```
    # move spacing mapping aprepro file up before cleaning dir
    cp space_mapping_vars ../.
    cd ..

    breaksw # dependent

case default:
    # move default_vars into space_mapping_vars

    cp ../templatedir_lowfi/default_vars ./space_mapping_vars

    breaksw #dependent

endsw # switch($ls_calc_type)

#---------------
# Optimize the Low Fidelity Model
#---------------
# move all low fidelity model files and space_mapping_vars
# into new workdir_opt
cp -r ../templatedir_opt workdir_opt
cp ../templatedir_lowfi/* workdir_opt
cp -r ../templatedir_lowfi workdir_opt/templatedir
cp space_mapping_vars workdir_opt/templatedir
cp space_mapping_vars workdir_opt/
cp low_fi_script workdir_opt/
cd workdir_opt


switch($opt_strategy)
case grad:

    # prepare optimization input file
    m4 -DNUM_DESIGN_VAR="$num_low_var" \\
      -DCDV_INITIAL="$low_var_init_string" \\
      -DCDV_UPPER="$low_var_high_string" \\
      -DCDV_LOWER="$low_var_low_string" \\
      -DCDV_TAG="$low_var_tag_wparen_string" \\
      dakota_lowfi_opt_template.in > dakota_lowfi_temp.in

    aprepro -qW dakota_lowfi_temp.in dakota_lowfi_opt.in

    # do low fidelity optimization with space mapping variables
    dakota -input dakota_lowfi_opt.in >>&! dakota_lowfi_opt.out
```

```
    #place low fidelity optimum result within file
    java ParseFileToAprepro dakota_tabular dakota_tabular.dat 0 \\
      lowfi_optimum_vars

    breaksw #grad
case apps:

    # append apps_run_script_template file to low_fi_script, so that
    # APPSPACK input file is appropriate aprepro format
    grep -v "#!" low_fi_script > script_temp

    m4 -DNUM_LOW_VAR="$num_low_var" \\
       -DLOW_VAR_TAGS="$low_var_tag_string" \\
       apps_run_script_template > apps_run_script

    cat apps_run_script script_temp > low_fi_script_new
    mv low_fi_script_new low_fi_script
    chmod a+x low_fi_script
    rm apps_run_script script_temp

    # prepare optimization input file
    m4 -DNUM_DESIGN_VAR="$num_low_var" \\
       -DINITIAL_X="$low_var_init_string" \\
      -DUPPER_BOUNDS="$low_var_high_string"  \\
      -DLOWER_BOUNDS="$low_var_low_string" \\
      appsinput_template.apps > appsinput_temp.apps

    aprepro -qW appsinput_temp.apps appsinput.apps

    # do low fidelity optimization with space mapping variables
    mpirun -np 2 \\
/home/jpcastr/APPSPACK/test/appspack-4.0alpha-dev/src/appspack_mpi \\
    appsinput.apps > appsinput.out

    #place low fidelity optimum result within file
    java ParseFileToAprepro apps_out appsinput.out \\
      $num_high_var $high_var_tag_string lowfi_optimum_vars \\
      $response_tag $number_responses

    breaksw #apps

endsw # switch($opt_strategy)

# move optimum  aprepro file up before cleaning up this directory
cp lowfi_optimum_vars ../.
```

```
cd ..

#---------------
# Write out to APPSPACK function output file
#---------------

# place all responses in a single file (response.txt)
@ iv = 0
while ($iv < $num_high_var)
    @ var_index = $iv + $start_high_index
    m4 -DVAR_FILE="lowfi_optimum_vars" \\
       -DVAR_NAME=$argv[$var_index] \\
       response_template.txt > temp_response_$iv.txt
    aprepro -qW temp_response_$iv.txt temp_response_$iv.app
    perl -ne '{print unless /^\s*$/o}' temp_response_$iv.app \\
      > temp_response_$iv.out
    cat temp_response_$iv.out >>! response.txt
    rm temp_response*.*
    @ iv++
end

#
# CREATE NEW ARGV{1} WITH NEW VALUES
#
echo $num_high_var > num_var.txt
echo $message > message.txt
cat num_var.txt response.txt message.txt > $argv[1]
cp $argv[1] ../.

cp space_mapping_vars ..

# move out of inner loop directory
cd ..

#-----------------
# create a matlab friendly file with space_mapping_vars
#-----------------

@ j=0
while ($j < $num_space_param)
    @ varindex = $j + $start_space_param_index
    m4 -DRUN_TAG="$num" -DSPACE_PARAM_TAG="$argv[$varindex]" \\
      matlab_template.in > temp_matlab.in
    aprepro -qW temp_matlab.in | perl -ne \\
      '{print unless /^\s*$/o}' > temp_matlab.out
```

```
    cat temp_matlab.out >> matlab_space_map_vars.m
    rm temp_matlab.in temp_matlab.out
    @ j++
end
rm space_mapping_vars

# make call to high fidelity model

high_fi_script $argv[1] $argv[2] $argv[3] $message $fidelity \\
  $num_high_var $high_var_tag_string

# -------------
# CLEANUP
# -------------

rm -rf innerloop.$num
```

END mfo_script

DISTRIBUTION:

1 Prof. Katie Fowler
Clarkson University
PO Box 5815
Potsdam, NY 13699-5815

1 Prof. John W. Bandler
Simulation Optimization Systems Research Laboratory
Department of Electrical and Computer Engineering
McMaster University
1280 Main Street West
Hamilton, Ontario, CANADA L8S 4K1

1 Prof. Karen Wilcox
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
77 Massachusetts Ave.
37-395
Cambridge, MA 02139-4307

1 Ms. Theresa Robinson
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
77 Massachusetts Ave.
37-395
Cambridge, MA 02139-4307

10 MS 0316
Joseph Castro, 1437

1 MS 0316
Gary Hennigan, 1437

1 MS 0316
Robert Hoekstra, 1437

1 MS 0316
Eric Keiter, 1437

1 MS 0316
Paul Lin, 1437

1 MS 0316
Larry Musson, 1437

1 MS 0316
Eric Rankin, 1437

1 MS 0316
Richard Schiek, 1437

1 MS 0316
Rod Schmidt, 1437

1 MS 0316
John Shadid, 1437

1 MS 0370
Bart van Bloemen Waanders, 1411

1 MS 0316
Bret Bader, 1416

1 MS 0316
Russell Hooper, 1416

1 MS 0316
Roger Pawlowski, 1416

1 MS 0316
Eric Phipps, 1416

1 MS 0321
Bill Camp, 1400

1 MS 1110
David Womble, 1410

1 MS 0316
Sudip Dosanjh, 1420

1 MS 0321
Jennifer Nelson, 1430

1 MS 0370
Scott Mitchell, 1411

1 MS 0310
Mark Rintoul, 1412

1 MS 1110
Scott Collis, 1414

1 MS 1110
Suzanne Rountree, 1415

1 MS 1110
Andrew Salinger, 1416

1 MS 0378
Randy Summers, 1431

1 MS 0378
Jim Strickland, 1433

1 MS 1110
John Aidun, 1435

1 MS 0316
Scott Hutchinson, 1437

5 MS 9159
Genetha Gray, 8962

1 MS 9159
Heidi Ammerlahn, 8962

1 MS 9159
Josh Griffin, 8962

1 MS 9159
Tammy Kolda, 8962

5 MS 9159
Patricia Hough, 8962

5 MS 0828
Anthony Giunta, 1533

1 MS 0370
Mike Eldred, 1411

1 MS 0370
Laura Swiler, 1411

1 MS 0370
David Gay, 1411

1 MS 0370
Shannon Brown, 1411

1 MS 0370
Timothy Trucano, 1411

1 MS 0370
Stephen Thomas, 1411

1 MS 0828
Vincente Romero, 1533

1 MS 0847
Walter Witkowski, 1533

1 MS 1110
William Hart, 1415

1 MS 1110
Jean-Paul Watson, 1415

2 MS 9018
Central Technical Files, 8945-1

2 MS 0899
Technical Library, 4536

1 MS 0123
D. Chavez, LDRD Office, 1011

Sandia National Laboratories