12-2011

Optimization of a Low Reynolds Number 2-D Inflatable Airfoil Section

Todd A. Johansen
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd
Part of the Aerospace Engineering Commons, and the Mechanical Engineering Commons

Recommended Citation
https://digitalcommons.usu.edu/etd/864

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact dylan.burns@usu.edu.
OPTIMIZATION OF A LOW REYNOLDS NUMBER 2-D INFLATABLE AIRFOIL SECTION

by

Todd A. Johansen

A thesis submitted in partial fulfillment of the requirements for the degree
of
MASTER OF SCIENCE
in
Mechanical Engineering

Approved:

Dr. Thomas Hauser  Dr. Heng Ban
Major Professor  Committee Member

Dr. Stephen Whitmore  Dr. Byron R. Burnham
Committee Member  Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah
2011
Abstract

Optimization of a Low Reynold’s Number 2-D Inflatable Airfoil Section

by

Todd A. Johansen, Master of Science
Utah State University, 2011

Major Professor: Dr. Thomas Hauser
Department: Mechanical and Aerospace Engineering

A stand-alone genetic algorithm (GA) and an surrogate-based optimization (SBO) combined with a GA were compared for accuracy and performance. Comparisons took place using the Ackley Function and Rastrigin’s Function, two functions with multiple local maxima and minima that could cause problems for more traditional optimization methods, such as a gradient-based method. The GA and SBO with GA were applied to the functions through a fortran interface and it was found that the SBO could use the same number of function evaluations as the GA and achieve at least 5 orders of magnitude greater accuracy through the use of surrogate evaluations.

The two optimization methods were used in conjunction with computational fluid dynamics (CFD) analysis to optimize the shape of a bumpy airfoil section. Results of optimization showed that the use of an SBO can save up to 553 hours of CPU time on 196 cores when compared to the GA through the use of surrogate evaluations. Results also show the SBO can achieve greater accuracy than the GA in a shorter amount of time, and the SBO can reduce the negative effects of noise in the simulation data while the GA cannot.
Acknowledgments

I would like to acknowledge the help of Dr. Thomas Hauser and Dr. Raymond LeBeau and thank them for offering their insight and support as I carried out my thesis research.

Todd A. Johansen
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Optimization Algorithms</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Genetic Algorithm</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 Surrogate-Based Optimization</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Inflatable Wings</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Problem Statement</td>
<td>3</td>
</tr>
<tr>
<td>2 Literature Review</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Genetic Algorithm</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Surrogate-Based Optimization</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Inflatable Wings</td>
<td>8</td>
</tr>
<tr>
<td>3 Genetic Algorithm with Surrogate Model</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Genetic Algorithm</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Surrogate-Based Optimization</td>
<td>13</td>
</tr>
<tr>
<td>3.3 DAKOTA</td>
<td>14</td>
</tr>
<tr>
<td>3.3.1 Genetic Algorithm in DAKOTA</td>
<td>15</td>
</tr>
<tr>
<td>3.3.2 Surrogate-Based Optimization in DAKOTA</td>
<td>15</td>
</tr>
<tr>
<td>4 Optimization of Test Functions</td>
<td>18</td>
</tr>
<tr>
<td>4.1 Ackley Function</td>
<td>18</td>
</tr>
<tr>
<td>4.2 Rastrigin’s Function</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Optimization results</td>
<td>23</td>
</tr>
<tr>
<td>5 Inflatable Airfoil Optimization</td>
<td>28</td>
</tr>
<tr>
<td>5.1 Computational Tools</td>
<td>28</td>
</tr>
<tr>
<td>5.1.1 OpenFOAM</td>
<td>28</td>
</tr>
<tr>
<td>5.1.2 Meshing</td>
<td>28</td>
</tr>
<tr>
<td>5.1.3 Parallel Execution</td>
<td>29</td>
</tr>
<tr>
<td>5.2 Interface Between OpenFOAM and DAKOTA</td>
<td>29</td>
</tr>
<tr>
<td>5.2.1 Optimization Variables</td>
<td>30</td>
</tr>
<tr>
<td>5.2.2 Sample Points</td>
<td>30</td>
</tr>
<tr>
<td>5.2.3 Airfoil Geometry</td>
<td>30</td>
</tr>
</tbody>
</table>
5.2.4 Computational Mesh ........................................... 32
5.2.5 Function Evaluations ........................................... 34
5.3 Airfoil Surrogate-Based Optimization with Genetic Algorithm .......... 37
  5.3.1 Surrogate-Based Optimization Setup ................................ 37
  5.3.2 Surrogate-Based Optimization Results .............................. 38
5.4 Airfoil Optimization with Genetic Algorithm ............................ 42
  5.4.1 Genetic Algorithm Setup ........................................... 43
  5.4.2 Genetic Algorithm Optimization Results ............................. 43
5.5 Comparison of Optimization Methods .................................... 43
6 Conclusions and Discussion .............................................. 47
  6.1 Surrogate Model With Genetic Algorithm ............................. 47
  6.2 Airfoil Performance .................................................... 47
  6.3 Flow Separation .......................................................... 48
  6.4 Summary and Future Work .............................................. 50
References ...................................................................... 52
Appendices .................................................................... 55
  Appendix A DAKOTA Ackley Function Test Input Files ............... 56
    A.1 DAKOTA input file for Ackley function surrogate-based optimimization. 56
    A.2 DAKOTA input file for Ackley function genetic algorithm. ........ 59
  Appendix B DAKOTA Airfoil Optimization Input Files ............... 61
    B.1 DAKOTA input file for airfoil surrogate-based optimimization. ...... 61
    B.2 DAKOTA input file for airfoil genetic algorithm. .................. 64
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Optimal Airfoils from Surrogate Optimization.</td>
<td>38</td>
</tr>
<tr>
<td>5.2</td>
<td>Best Airfoil from Genetic Algorithm Optimization.</td>
<td>43</td>
</tr>
<tr>
<td>5.3</td>
<td>Best Airfoils from OpenFOAM Simulations in Surrogate Optimization.</td>
<td>44</td>
</tr>
<tr>
<td>5.4</td>
<td>Comparison of Run Times for SBO and GA.</td>
<td>44</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Smooth and Bumpy Airfoil Sections.</td>
<td>4</td>
</tr>
<tr>
<td>3.1 Genetic Algorithm Optimization Procedure.</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Surrogate-Based Optimization Procedure.</td>
<td>14</td>
</tr>
<tr>
<td>3.3 Parallelism Employed by DAKOTA.</td>
<td>15</td>
</tr>
<tr>
<td>4.1 Plot of Ackley Function for a Single Variable.</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Function Value vs Crossover Rate.</td>
<td>20</td>
</tr>
<tr>
<td>4.3 Optimization Time vs Crossover Rate.</td>
<td>20</td>
</tr>
<tr>
<td>4.4 Function Value vs Mutation Rate.</td>
<td>22</td>
</tr>
<tr>
<td>4.5 Optimization Time vs Mutation Rate.</td>
<td>22</td>
</tr>
<tr>
<td>4.6 Plot of Rastrigin’s Function for a Single Variable.</td>
<td>23</td>
</tr>
<tr>
<td>4.7 Function Value Compared to Optimization Time.</td>
<td>24</td>
</tr>
<tr>
<td>4.8 Function Value vs the Number of Function Evaluations.</td>
<td>25</td>
</tr>
<tr>
<td>4.9 Optimization Time Compared to Function Evaluations.</td>
<td>27</td>
</tr>
<tr>
<td>5.1 Surrogate-Based Optimization Procedure for the Airfoil Optimization.</td>
<td>30</td>
</tr>
<tr>
<td>5.2 Points Used to Create Modifiable Airfoils.</td>
<td>31</td>
</tr>
<tr>
<td>5.3 Individual Bump on Airfoil Surface.</td>
<td>32</td>
</tr>
<tr>
<td>5.4 Example of Clustering Applied to Bumpy Airfoil.</td>
<td>33</td>
</tr>
<tr>
<td>5.5 Possible Airfoils Created During Optimization.</td>
<td>33</td>
</tr>
<tr>
<td>5.6 Example of Unstructured Computational Mesh Created for Optimization.</td>
<td>35</td>
</tr>
<tr>
<td>5.7 Cl/Cd Data from Optimization at $\alpha = 2$.</td>
<td>39</td>
</tr>
<tr>
<td>5.8 Trendlines of Cl/Cd from Optimization at $\alpha = 2$.</td>
<td>40</td>
</tr>
</tbody>
</table>
5.9 Cl/Cd Data from Optimization at $\alpha = 5$. . . . . . . . . . . . . . . . . . . . 40
5.10 Trendlines of Cl/Cd from Optimization at $\alpha = 5$. . . . . . . . . . . . . . . 41
5.11 Cl/Cd Data from Optimization at $\alpha = 10$. . . . . . . . . . . . . . . . . . . . 41
5.12 Trendlines of Cl/Cd from Optimization at $\alpha = 10$. . . . . . . . . . . . . . 42
5.13 Comparison of Lift-to-Drag Ratio vs Number of FOAM Simulations. . . . . 46
6.1 Plots of Cl/Cd vs $\alpha$ for 1st and 2nd Order Results. . . . . . . . . . . . . . . 49
6.2 Plots of Cl vs $\alpha$ for 1st and 2nd Order Results. . . . . . . . . . . . . . . . 49
6.3 Plots of Cd vs $\alpha$ for 1st and 2nd Order Results. . . . . . . . . . . . . . . . 49
6.4 Velocity Profiles for the Original and 2º Optimized Airfoils at $\alpha = 2$. . . . 50
6.5 Velocity Profiles for the Original and 10º Optimized airfoils at $\alpha = 5$. . . . 50
Chapter 1

Introduction

This research is a comparison of optimization algorithms on a computationally expensive target. The algorithms compared are a genetic algorithm and a surrogate-based optimization method coupled with a genetic algorithm. The methods were first applied to test functions evaluated by a simple fortran program to determine the ideal setup for a more complicated optimization.

The second optimization problem the methods were applied to is the optimization of a bumpy airfoil section used to construct an inflatable wing. The airfoil section is based on the Eppler 398 baseline profile. The number of bumps on the surface of the airfoil as well as the radii of the bumps were changed to alter performance. Performance was measured by the lift-to-drag ratio returned from computational fluid dynamics (CFD) simulations. Running these simulations is prohibitively expensive so it is desirable to reduce the necessary CFD simulations required for a satisfactory result. The algorithms were compared to determine whether the surrogate-based optimization could reduce computational expense through evaluations of the surrogate model while maintaining accuracy.

1.1 Optimization Algorithms

1.1.1 Genetic Algorithm

Genetic algorithms are a fairly recent but popular breakthrough in optimization. They are based on principles of evolutionary biology and use concepts such as fitness, mutation, inheritance, crossover, and selection. Genetic algorithms use a global search method where multiple sample points can be tested simultaneously. Fitness determines whether sample points will be used to propagate new sample points. The propagation of points can be
carried out by mutating existing points or combining existing points.

Benefits of genetic algorithms include the use of stochastic methods that allow for easy parallelization, they can handle multiple objective functions, and they can be easily applied to complex engineering problems and linked with a variety of evaluation programs.

1.1.2 Surrogate-Based Optimization

Surrogate models can be used to approximate a model using the results of sample points from the original model. The surrogate model can then be optimized instead of the actual model, which is beneficial for optimization studies where evaluations of the original model incur high computational cost, such as in a CFD simulation.

Several types of surrogate formulations are available, including data fits, multifidelity models, and reduced-order models. Data fit models use non-physics based approximation involving interpolation of sample data points. A polynomial data fit model is used in this study in an attempt to reduce the number of CFD simulations required.

A benefit of using a global data fit model is that this model can reduce the negative effects of poorly-behaved, nonsmooth, discontinuous response variations within the original model. This allows the global surrogate model to extract relevant information from noisy simulation data. Global data fit models do not guarantee convergence however. The noise from simulations can limit the accuracy of a global model.

Local surrogate data fit models use a trust region approach where the range of variables is adjusted according to the accuracy of the constructed model. Noisy simulation data will cause a reduction in the size of the trust region in order to improve accuracy. This approach may not necessarily converge to a result where the trust region stops changing but it will still have the benefit of reduced noise, allowing design trends to be extracted from simulation results. Given the ability to limit the effects of noise from simulations and to reduce the use of computationally expensive simulation evaluations, surrogate models can speed up the rate of convergence.
1.2 Inflatable Wings

Unmanned air vehicles (UAVs) are seeing increased use with various sizes and possible applications. Some UAVs are powered by electric motors and are small enough to carry and deploy by hand while other UAVs are powered by jet engines, are as large as conventional aircraft, and are capable of delivering bombs and missiles in military attacks. UAVs can also be used for reconnaissance, research and development, and scientific studies. The wings of some UAVs are capable of folding into smaller configurations for easy transport. Research and development of UAVs has led to the development of inflatable wings that can be packaged compactly and easily deployed.

Investigation of inflatable wings at the University of Kentucky has its roots in the BIG BLUE project, which was a senior design project for students in mechanical and electrical engineering. As part of the project several different inflatable wing configurations were tested that were based on the Eppler 398 and NACA4318 airfoil sections [1]. As a result of manufacturing techniques the surface of the inflatable wings are bumpy. The bumps provide lateral strength in the wings but can reduce performance. The profiles of the smooth and bumpy Eppler 398 airfoils are displayed in figure 1.1.

1.3 Problem Statement

The objective of this research is to compare a genetic algorithm with a surrogate-based optimization combined with a genetic algorithm. They are compared for accuracy as well as time to optimum solution to determine whether the surrogate-based optimization can outperform the genetic algorithm. As part of the research the lift-to-drag ratio of a bumpy airfoil section will be optimized by both algorithms at a Reynolds number of 25,000. The lift to drag ratio is altered by controlling the number and size of the bumps on the upper and lower surfaces of the airfoil. The goal of this comparison is to show that the surrogate-based optimization can obtain better results while lowering computational expense.
Fig. 1.1: Smooth and Bumpy Airfoil Sections.
Chapter 2

Literature Review

2.1 Genetic Algorithm

A survey of literature for genetic algorithms focused on the various uses for genetic algorithms. They range from economics and resource planning problems to general engineering and aerospace engineering applications. Some of the various uses of a genetic algorithm include the cutting stock problem, allocation of railway platforms, airline crew scheduling, mine ventilation, and power generation expansion.

A genetic algorithm was used to study the one-dimensional cutting stock problem, which is designed to reduce the amount of construction waste generated. Real life case studies of steel mills were found in which genetic algorithms showed a high potential of waste savings that could be achieved. That waste savings in turn could increase profits and reduce environmental impact [2].

Clark et al. have applied a genetic algorithm to the ideal allocation of a railway station’s resources. The approach was able to successfully allocate roughly 1000 trains to platforms while meeting the needs of the station schedule [3].

Another allocation problem which makes use of genetic algorithms is airline crew scheduling. The goal in solving this problem is to assign sets of crew members to schedules that minimize cost without causing too much disruption to crew members. By analyzing the schedule of an actual airline Kornilakis and Stamatopoulos showed that a genetic algorithm could be used to improve the cost of current scheduling [4].

Park et al. have developed an improved genetic algorithm for application to a least-cost generation expansion problem. This problem is concerned with the ideal expansion of a power grid due to growth. The problem can only be fully solved by enumerating all possibilities of existing power plants, candidate power plants, and planning periods. This is
impossible in real world application but they have shown that their genetic algorithm can be a successful tool for long term planning [5].

Genetic algorithms can be especially useful in engineering problems. Ahuactzin et al. have applied a genetic algorithm to the problem of robot motion planning with the goal of building fast motion planners for robots with six or more degrees of freedom. Initial tests show that the genetic algorithm was able to make fast response planning possible for a robotic arm with two degrees of freedom [6].

Axial compressor rotor blades have been optimized by a genetic algorithm using a multiobjective approach with total pressure and adiabatic efficiency as objective functions in CFD simulations. The genetic algorithm was able to successfully increase total pressure and maximum efficiency by 0.41% and 1.76% respectively [7].

Makinen, Periaux, and Toivanen applied a genetic algorithm to a multiobjective multidisciplinary design optimization of a two-dimensional airfoil. The approach was designed to minimize the airfoil drag at a specific lift coefficient as well as radar visibility of the airfoil by combining a CFD solution of the inviscid Euler equations with the solution of a two-dimensional Helmholtz equation [8].

Holst and Pulliam demonstrated an approach for optimizing the aerodynamic shape of a transonic wing by coupling a genetic algorithm with a transonic potential flow solver. The genetic algorithm had control over 10 parameters used to alter the shape of the airfoil. The results of their study illustrate the flexibility and reliability of a genetic algorithm [9].

Sasaki et al. applied a genetic algorithm to the multiobjective optimization of a wing in transonic and supersonic flow regimes. Design objectives were to minimize transonic and supersonic drag coefficients and the bending and twisting moments of the wings in supersonic flight. Results confirmed the algorithm works well in a large search space and can improve upon current design [10].

Thiesinger and Braun tested methods of optimization of hypersonic entry aeroshell shapes. A genetic algorithm is used to optimize the possible curve formulations for the blunt leading edge of the re-entry vehicle using MATLAB routines that perform shape
2.2 Surrogate-Based Optimization

A review of literature involving the use of surrogate-based optimization focuses on uses similar to those of the research in this paper and on the effectiveness of such optimization methods. The studies discussed are typically ones involving the combination of a surrogate model with CFD simulations, resulting in a reduction of computational expense.

Eldred et al. present the results of research performed at Sandia National Laboratories which tests several formulations for optimization under uncertainty. The surrogate formulations encompass data fit and hierarchical surrogate models. Testing with two analytic problems and one engineering problem is used to compare merits of different methodologies. Results show that surrogate-based optimization under uncertainty formulations show promise in reducing the number of function evaluations required and in reducing the effects of non-smooth response variations. Furthermore, weaknesses in data fits can lead to poor solutions but trust-region approaches to surrogate-based optimization under uncertainty can maintain the quality of results [12].

Jansson, Nilsson, and Redhe have shown that a surrogate approach can be used for optimization in automobile crash worthiness design and sheet metal forming applications with a significant reduction in computing time compared to conventional optimizations. Results also show that a surrogate approach can converge to an improved crashworthiness design, even in cases where conventional optimizers failed to converge [13].

A surrogate-based approach was used in the optimization of a four parameter design experiment using CFD. The objective of the research was to increase the efficiency of a scramjet fuel injector. No optimum geometry was found as the results of the study showed the need to explore different injector geometries outside of the current setup [14].

Riesenthal and Childs applied a surrogate-based optimization approach to the design problem of an asymmetric fairing for a launch vehicle. Design goals were to balance a low lateral force on the fairing and smooth variations in that force with respect to angle of attack across a range of Mach numbers near Mach 1.0. After 10 design iterations, which
amounted to less than 230 geometric configurations, a large number of acceptable designs were found [15].

Glaz, Friedmann, and Liu studied the use of various surrogates in modeling vibrations of helicopter rotor blades in forward flight. The accuracies of kriging, radial basis function interpolation, and polynomial regression surrogates are compared. The optimized blade was compared with a baseline rotor blade resembling an MBB BO-105 blade. Results show that kriging surrogates were the best method for approximation of vibration loads and that the surrogates can be effective in studies of helicopter rotor vibration reduction [16].

In a review of the Optimization and Uncertainty Estimation department at Sandia National Labs, Eldred discussed the successful application of a surrogate-based optimization of drop tanks for use on the F-35 joint strike fighter performed by Lockheed Martin Aeronautics. The tanks were optimized for minimum drag and a minimum yawing moment for separation of stores adjacent to the tanks. Data obtained during the course of the optimization showed excellent agreement with wind tunnel experiments [17].

2.3 Inflatable Wings

Lightweight UAVs tested with inflatable wings operated in flight regimes with Reynolds numbers $O(10^4 - 10^5)$. A problem that results from a low Reynolds number flow is the formation of a separation bubble on the upper surface of the wing. This separation bubble causes the boundary layer to detach from the surface of the wing and causes an increase in pressure drag [18]. Depending on the configuration it is possible for the flow to reattach and form a turbulent boundary layer or remain unattached [19]. It has been shown that for smooth airfoils, boundary layer reattachment can occur above a Reynolds number of $7.0 \times 10^5$ [18].

Research included wind tunnel testing, computational fluid dynamics simulations, and flight testing. Santhanakrishnan and Jacob investigated the effects of introducing large-scale roughness through static curvature modifications on the low speed flow over an airfoil. Their smoke wire visualization results show that for the smooth Eppler 398 airfoil at $Re = 2.5 \times 10^4$ and $\alpha = 0$, separation starts close to the leading edge with no reattachment. The streamlines
indicate laminar flow prior to separation. Inflatable wing streamlines indicate that the flow is “tripped” by the bumps, promoting transition, but the flow does not reattach. However, the separation point was observed to be farther downstream for the bumpy wing than the smooth. The bumpy wing also exhibited a smaller separation bubble than the smooth wing [19].

Numerical studies of inflatable wings originated as part of discussions in the BIG BLUE project. The question was whether the inflatable bumpy wings needed a smooth skin over the bumps as well as the addition of a sharper trailing edge. The wind tunnel experiments performed by Santhanakrishnan helped lead to the conclusion that adding a covering over the bumps was unnecessary due to improvements in the flow due to the bumps [1].

Reasor and LeBeau compared two-dimensional CFD simulations of a smooth Eppler 398 wing profile with those of an inflatable Eppler 398 wing profile. Their results agree with experimental studies which suggest that the bumpy airfoil experiences less flow separation than the smooth airfoil for low Reynolds numbers. They also show that flow is more unsteady over the bumpy airfoil over a range of angles of attack, leading to an increase in drag and a decrease in lift [1].

LeBeau et al. have investigated the effects of a bumpy surface on two different inflatable airfoils. One is based on the Eppler 398 while the other is based on the NACA 4318 airfoil. Two-dimensional CFD simulations have shown that bumpiness has a considerable effect on performance but the effects appear to depend significantly on the baseline airfoil shape [20].

Smith et al. have conducted numerous flight experiments of the inflatable wing aircraft. They have also been successful in high altitude deployment tests of inflatable wings. The inflatable wing aircraft was carried to a high altitude by a balloon where the wings were deployed. The packaged wings were restrained but pressurization caused the restraints to release, allowing the wings to unfold. One of the high altitude experiments resulted in the loss of a wing as a restraint held longer than expected and the increase in pressure during inflation caused a sudden release of the wing, which led to damage of the fuselage and loss of the wing, preventing the planned free flight. The culmination of flight testing was a
successful low altitude autonomous flight of the inflatable wing aircraft [21].

Additional research in inflatable wing technology is the use of wing morphing as a method of roll control. Cadogan et al. have performed extensive studies of different methods of trailing edge deflection. Results indicate that the use of trailing edge piezo actuators were a suitable method of achieving roll control with a rapid response rate. One benefit of this method was the production of an adequate rolling moment by a surface with no defined hinge or airfoil discontinuity, resulting in increased efficiency over traditional roll control methods. Other methods such as bump flattening and the use of nastics technology show promise given improvement upon current technology used to implement the methods [22].
Chapter 3
Genetic Algorithm with Surrogate Model

3.1 Genetic Algorithm

A genetic algorithm is a search methodology which mimics the process of evolution. Solutions to optimization problems are generated using concepts such as inheritance, natural selection, mutation, and genetic crossover. Sample points within the population will evolve from randomly generated individuals toward an ideal solution. The algorithm performs an optimization by sampling points then determining which samples return the most optimal values. The most optimal, or "fit," points will be retained and allowed to propagate a new generation of test points in their vicinity. The new points will be evaluated and again the most fit points will be retained. This process will continue until a termination criteria is set. This method can be used on its own or as part of another optimization approach. The basic algorithm is outlined in figure 3.1.

![Genetic Algorithm Flowchart](image)

Fig. 3.1: Genetic Algorithm Optimization Procedure.

The evolution of the genetic algorithm in this study can be controlled by changing the following variables in the input file for the genetic algorithm:
1. Crossover_rate: This controls the probability of combining "parent" genetic information to create offspring (a new sample point) with 1.0 being 100% probability and 0.0 being 0% probability.

2. Mutation_rate: This controls the probability of mutation taking place on an individual point. The mutation of a point is simply a change in the point’s location.

3. Fitness_type: This controls how the difference in "fitness" of data points affects the process of selecting parent points for the next iteration of the GA. The probability of selecting a point can be scaled based on the rank of its function value compared to the rest of the population or by scaling the probability proportionally based on the relative values of function evaluations within the population.

4. Crossover_type: This determines how parent information is combined to create offspring. Spatial coordinates can be taken from two parents and used as coordinates for offspring, a new point can be created on a vector connecting two parent points, or the coordinates from two parent points can be randomly combined to create a new point.

5. Replacement_type: This controls how current points and newly generated points combine to create a new population of points. New points can be randomly created from the entire population, from a combination of the best points in the entire population and the latest iteration, a combination of the best points from the population and random points from the current population, or exclusively the best points from the current iteration.

6. Initialization_type: This defines how the population of the algorithm is initialized. The population can be read in from a file or created randomly, with specifications to ensure that there are no duplicate sample points in the initial population.

7. Mutation_type: This controls the approach used to modify continuous design variables. Some approaches generate random replacements for coordinates while some
scale the coordinates, and still others use a uniform offset of each coordinate to create a new point.

The genetic algorithm in DAKOTA follows the approach outlined in figure 3.1. To generate offspring the algorithm uses crossover from two selected parents with a fixed probability, then applies mutation to the new individuals with a fixed probability. If crossover is not used a single parent is allowed to mutate according to a fixed probability.

3.2 Surrogate-Based Optimization

In a surrogate-based optimization the optimization algorithm, such as a genetic algorithm, operates directly on a computationally inexpensive surrogate model instead of computationally expensive simulation results. The surrogate-based optimization evaluates initial sample points in the test region then creates a model of the results. The surrogate model is then optimized. New sample points are generated, a new model is created, then optimized again. This process continues for a user specified number of iterations. To illustrate the process, figure 3.2 is included. There are multiple categories to choose from for surrogate models, such as data fits, multifidelity models, and reduced-order models. For its simplicity, the data fit model approach was selected for this optimization.

The surrogate method used in this study is a surrogate-based local minimization approach. This is an approach in which the variable domain of an iteration, called the trust region, is allowed to change depending upon the accuracy of the surrogate model. When the optimum value of a surrogate model iteration is found, the parameters for the optimal point are evaluated in the actual model and compared to the surrogate value. If these values have excellent agreement, the surrogate model has excellent accuracy and the current trust region is allowed to expand to include a greater range of variables. If the surrogate model has poor accuracy as determined by a poor comparison between the values, then the trust region will contract in an attempt to increase accuracy. Instances can also occur in which the accuracy is deemed ”acceptable” and the trust region will remain the same for the next iteration of the surrogate model.
3.3 DAKOTA

The Design Analysis Kit for Optimization and Terascale Applications (DAKOTA) toolkit from Sandia National Laboratories provides an interface between simulation codes and analysis methods [23]. DAKOTA has numerous optimization schemes which include algorithms for gradient and non-gradient based optimizations. DAKOTA also contains algorithms for uncertainty quantification with sampling, reliability, polynomial chaos, and epistemic methods; parameter estimation with nonlinear least squares methods; and sensitivity/variance analysis with design of experiment methods and parameter study methods.

DAKOTA is designed for use with large scale problems, so capabilities have been included for DAKOTA to operate with multiple levels of parallelism. A single level parallel approach is to create a master communicator that distributes work among parallel processors (fig. 3.3(a)). An approach with multiple levels of parallelism creates a master communicator that is partitioned into slave communicators. The slave communicators in turn distribute work among groups of parallel processors or they can be partitioned into even more slave
communicators (fig. 3.3(b)). The testing process of DAKOTA employed a single level of parallelism and the airfoil optimization employed two levels of parallelism.

(a) Single Level Parallelism.  
(b) Multi-Level Parallelism.

Fig. 3.3: Parallelism Employed by DAKOTA.

3.3.1 Genetic Algorithm in DAKOTA

The genetic algorithm used in this study is the coliny_ea algorithm contained in DAKOTA. The basic control variables for the genetic algorithm are population size, max iterations, and max function evaluations. The population size is the number of sample points generated for each iteration of the GA. The parameters max_iterations and max_function_evaluations control when the algorithm stops. If the maximum number of iterations for the algorithm is reached before the maximum number of function evaluations then the algorithm will terminate. Similarly, reaching the maximum number of function evaluations will terminate the optimization.

3.3.2 Surrogate-Based Optimization in DAKOTA

The surrogate-based optimization in DAKOTA is a surrogate-based local minimization using a quadratic polynomial as the surrogate model and combined with the coliny_ea genetic algorithm as the optimization algorithm.

The DAKOTA input file for the surrogate-based optimization is split into sections for creating the surrogate model, optimizing the surrogate model, and for the overall control of the optimization. The sections are as follows:

1. Strategy
The strategy section allows the user to specify the use of a single or multiple optimization methods. The user can also specify the creation of a tabulated results output file.

2. Method

The method section allows the user to select a local or global surrogate model. The local surrogate model uses a variable trust region approach while a global surrogate approach maintains the initial bounds through the entire optimization. This section specifies the number of iterations of the surrogate model as well as a soft convergence limit. The soft convergence limit is reached after a certain number of iterations return identical solutions, terminating the optimization. The method section also allows the user to control the initial size of the trust region along with how the trust region varies throughout the optimization.

3. Optimizer method

The optimizer method section specifies the method used to optimize each iteration of the surrogate model.

4. Model

The optimizer model section specifies the data fit model used in the optimization. The global data fit models include a neural network, kriging interpolation, moving least squares, radial basis, and quadratic polynomial. A local Taylor series can also be used as a data fit model. The model used for testing and the airfoil optimization is a global quadratic polynomial.

5. Responses

The responses section specifies the number of objective functions to optimize. It also specifies what type of gradients and hessians to use in the creation of the surrogate model.

6. Sampling method
The sampling method specifies how the data points will be generated. The number of sample points to use in the construction of the surrogate model are also specified in this section.

7. Interface

This section specifies the simulation interface and the parallelism of DAKOTA. The simulation interface is determined by the analysis driver, parameters file, and results files specified.
Chapter 4
Optimization of Test Functions

Two test functions were used to compare possible configurations for the GA and the SBO combined with the GA. They are the Ackley function and Rastrigin’s function. These two functions contain multiple local maxima and minima making them an ideal comparison of non-gradient based optimization methods.

A fortran program was written to evaluate the test functions. It read in the variable values through a simulator script that acted as a an interface between DAKOTA and the fortran code. The simulator script then read the fortran output and returned the function values to DAKOTA. The GA repeatedly called the fortran interface for its evaluations. The SBO called the interface as well as performed evaluations on the surrogate model built within DAKOTA.

4.1 Ackley Function

The Ackley function is:

\[
f(x_1, x_2) = -c_1 \ast \exp(-c_2 \ast \sqrt{\frac{1}{2} \sum_{i=1}^{2} x_i^2}) - \exp(\frac{1}{2} \sum_{i=1}^{2} \cos(c_3 x_i)) + c_1 + e
\]

\[x_i \in [-5.0, 5.0], \ i = 1, 2\]

\[c_1 = 20, \ c_2 = 0.2, \ c_3 = 2\pi, \ e = 2.71282\]

\[with \ min: \ f(\overline{x}) = 0 \ at \ \overline{x} = 0 \quad (4.1)\]

The Ackley function was used to test how crossover and mutation affected the results returned by the genetic algorithm. The crossover rate was varied from 0 to 1 in increments of 0.25 for each crossover type. The results of crossover testing are displayed in figures 4.2
Fig. 4.1: Plot of Ackley Function for a Single Variable.

Overall, the results are better with low crossover rates for all of the crossover types. The best minimum is found at a crossover rate of 0.0 for the blend and uniform crossover types while the best minimum is at 0.25 for the 2-point crossover type. For the 2-point and blend crossover types, changing the crossover rate has little effect on the time taken to find a solution. Changing the crossover rate for the crossover types only varies the time by up to two seconds. The uniform crossover type caused slightly more variation in the optimization time, changing by six seconds from the shortest time to the longest. The uniform crossover type with a crossover rate of 0.0 led to the shortest simulation time with the same function value as the other crossover types. Simulation time is a major concern for the airfoil optimization so it would appear that using this crossover setup is ideal.

The Ackley function was also used to test how the mutation type and mutation rate affected solutions. There were four mutation types tested: offset_cauchy, offset_normal, offset_uniform, and replace_uniform. The mutation rate, which is the probability of muta-
Fig. 4.2: Function Value vs Crossover Rate.

Fig. 4.3: Optimization Time vs Crossover Rate.
tion, was varied from 0.0 to 1.0 in increments of 0.25. The results of mutation testing are displayed in figures 4.4 and 4.5.

The mutation types have similar simulation times for all mutation rates tested. As the mutation rate increases the simulation time also increases for all mutation types (fig. 4.5). This is so because mutation is what creates new sample points. If there is a low mutation rate new points will not be created. If new points are not created the optimization will repeatedly try to evaluate the previous points. However, DAKOTA detects the duplicate and prevents the analysis from evaluating the sample points in the fortran function. Since the fortran function is not invoked the optimization takes much less time than an optimization with a 100 percent chance of mutation.

Since a low rate of mutation does not allow as many new sample points as a higher rate, the optimization is limited in accuracy, as seen in figure 4.4. For all mutation types with a mutation rate of 0.0 the best function value found is 6.3 and one of the parameters has a value of 1.619, which is a very poor solution. Changing the mutation types creates slightly different results, but as the mutation rate increases the best function value found for each mutation type gets closer to zero the true optimum value. For most of the test cases the replace_uniform mutation type results in the best solution. Using a mutation of rate of 1.0 (a 100 percent probability of mutation) results in the best solution.

The simulation time increases by a factor of ten from a mutation rate of 0.0 to 1.0, but even though lowering optimization time is important, the accuracy of the optimization is limited too much by not allowing mutation. Based on this reasoning the best setup for mutation is to use the replace_uniform specification with a mutation_rate specification of 1.0.

4.2 Rastrigin’s Function

Alongside the Ackley function, another function was used to test how the number of sample points affected the optimization results in order to determine the ideal setup for the airfoil optimization. The function is Rastrigin’s function:
Fig. 4.4: Function Value vs Mutation Rate.

Fig. 4.5: Optimization Time vs Mutation Rate.
\[ f(\overline{x}) = \frac{1}{n} \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10], \quad n = 5 \]

\[ x_i \in [-0.12, 0.12], \quad i = 1, 2, 3 \]

\[ x_i \in [-1.12, 1.12], \quad i = 4, 5 \]

\textit{with min} : \quad f(\overline{x}) = 0 \text{ at } \overline{x} = 0 \quad (4.2)

Fig. 4.6: Plot of Rastrigin’s Function for a Single Variable.

4.3 Optimization results

Multiple cases were run on the Ackley function and Rastrigin’s function with the GA and SBO in which the number of function evaluations, the population size, and number of optimizer iterations were changed. For the genetic algorithm by itself and the genetic algorithm with the surrogate model the mutation and crossover settings were those described in the previous section. The results from the Ackley function and Rastrigin’s function are
very similar, so to avoid redundant plots, results from the Ackley function are omitted. Input files from test cases are included in appendix A.

Results from optimizations of Rastrigin’s function are included in figures 4.7, 4.8, and 4.9. From figure 4.7 it is clear that the surrogate-based optimization is able to achieve greater accuracy for the function value than the genetic algorithm in the same time span. After only 50 seconds the SBO has accuracy on the order of $10^{-6}$ while the GA only has accuracy on the order of $10^{-2}$. At 100 seconds the SBO has a function value of less than $10^{-12}$ while the GA only managed a function value slightly above 0.1. The GA is unable to match the accuracy of the SBO until the very last sample point, taking roughly 40 seconds longer to come to the same solution. These results indicate the SBO is not only much more accurate than the GA, but it is also more consistent. The SBO is more likely to increase in accuracy as it is allowed to run longer.

![Fig. 4.7: Function Value Compared to Optimization Time.](image)

Figure 4.8 compares the function value to the number of fortran function evaluations used. This comparison also shows the greater performance of the SBO over the GA. For a
given number of function evaluations, the SBO has much greater accuracy. The first point on the plot for the GA used 805 sample points to achieve a result just higher than 0.2. The first point for the SBO used 841 sample points to achieve a result just over $4.0 \times 10^{-5}$. Using roughly the same number of function evaluations the SBO was five orders of magnitude more accurate.

The most accurate result from the GA had a function value of $4.512 \times 10^{-14}$. The optimization corresponding to this point used 8005 function evaluations. The most accurate result from the SBO has a function value of $3.197 \times 10^{-14}$ while only using 1681 function evaluations, roughly 5 times less than the most accurate GA optimization. Clearly from these results, both optimization methods are likely to return a more accurate solution when the number of function evaluations are increased, but the SBO will see more of an improvement than the GA without increasing function evaluations as much.

The reason the SBO is able to increase accuracy without increasing the fortran function evaluations as much as the GA is through the surrogate function evaluations. The benefits of
increasing the surrogate evaluations while keeping the fortran function evaluations constant can be seen in figure 4.8. The first few points for the SBO create a vertical line in the plot. They all used 841 fortran function evaluations but are able to increase accuracy through the use of the surrogate optimizations. Within this group of points that create a vertical line, the first point used 32,240 surrogate evaluations while the final point used 121,840 surrogate evaluations.

The next group of points that form a vertical line in the SBO plot in 4.8 used 1,681 interface evaluations while the range of surrogate evaluations was from 320,480 to 480,480. The final result from the surrogate model used 2,521 interface evaluations and 960,720 surrogate evaluations.

Increasing the number of surrogate evaluations increases the run time of the surrogate model, as the points are not clustered in vertical lines in figure 4.7. Even with this increase, the time it takes to find an optimum value is still less than the time it takes the GA to converge to an equally accurate value. The surrogate model manages 2,521 interface evaluations and nearly 1 million surrogate evaluations in less time than it takes for the genetic algorithm to run through 8000 interface evaluations and return a function value on the same order of accuracy.

Figure 4.9 compares the optimization time to the number of function evaluations. The vertical lines in the plot better show how increasing the surrogate evaluations while keeping the fortran function evaluations constant can effect the optimization time. It is remarkable that increasing the number of surrogate evaluations by almost 90,000 sample points takes less time than increasing the number of GA fortran function evaluations by only 400 sample points. This is possible because the fortran program is external to DAKOTA while the surrogate model is built within DAKOTA. Outputting variables from DAKOTA into a format a fortran program can read, initializing the fortran program, writing the fortran output in a format DAKOTA can read, and reading those variables into DAKOTA creates quite a bit of overhead that does not exist for the surrogate model. This simple example illustrates the value a surrogate-based optimization will have when using computationally
expensive functions. If it can save time using a fortran function as the function call it should definitely be able to save time when using a more expensive function, such as a computational fluid dynamics simulation.

The interface for the DAKOTA tests is a simple fortran program that can return many values in a very short time. The interface for the airfoil optimization is a computational fluid dynamics simulation which takes roughly four hours to return a single lift-to-drag value. Using the surrogate-based optimization strategy for the airfoil and using the same configuration as the final Rastrigin’s function test has the potential to save up to 22,000 hours of computation time over the use of the genetic algorithm by itself. Due to limited time and resources, however, the previous setup for the number of function evaluations and surrogate evaluations cannot be used for the airfoil optimization. Even with the limitations the surrogate-based optimization should be able to increase the accuracy of the optimization solution while reducing the necessary number of CFD simulations.
Chapter 5
Inflatable Airfoil Optimization

5.1 Computational Tools

5.1.1 OpenFOAM

The CFD package used in the optimization is OpenFOAM. OpenFOAM is a free, open source software package that provides a variety of solvers and utilities which are designed to simulate problems in solid and fluid mechanics [24]. The OpenFOAM solver used in this optimization is icoFoam from OpenFOAM-1.5.x, which solves the incompressible Navier-Stokes equations using the Pressure Implicit with Splitting of Operators (PISO) algorithm. A first order upwind discretization scheme was used for the optimizations to save compute time. A second order central differencing scheme was applied to the optimized airfoils to test the accuracy of the solutions.

5.1.2 Meshing

Computational meshes used in this study were generated in Gmsh. Gmsh is a free, open source three dimensional finite element mesh generation tool with built in pre- and post-processing capabilities [25]. Gmsh provides simple parametric tools for meshing which are very useful for generating multiple geometries with only slight changes to the geometry file. Gmsh is ideal for this optimization because the airfoil can be altered by changing two parameters and OpenFOAM version 1.4.1 contains a utility called gmsh2ToFoam which creates the mesh directly from a Gmsh geometry file then converts that mesh into OpenFOAM format. This reduces the number of steps necessary to set up each case. It is also beneficial since gmsh2ToFoam retains all of the patch information of the mesh while other mesh converters available do not, saving the possible step of having to redefine the patch.
names in the boundary file once the mesh is created and converted.

5.1.3 Parallel Execution

OpenFOAM is capable of distributing computational work from a single simulation across multiple networked processors, which can drastically reduce the time a simulation takes to complete when compared to simulation time on a single processor. Benchmarks were performed with varying numbers of processors for a set mesh size and it was found that simulations completed in the shortest time possible when 16 processors were used for each simulation.

The airfoil optimization was carried out on the Wasatch cluster at the Utah State University Center for High Performance Computing. Wasatch is comprised of 64 compute nodes with dual quad-core AMD processors rated at 2.3 GHz with 8 GB memory per node with the nodes using an Infiniband interconnect. The surrogate optimization was run from the login node in parallel using 12 instances of DAKOTA and the genetic algorithm optimization ran using 8 instances of DAKOTA. Each instance of DAKOTA submitted a single OpenFOAM simulation to the job queue, with each job utilizing 16 processors. The maximum number of processors used at any given time was 196 out of the 512 available on Wasatch.

5.2 Interface Between OpenFOAM and DAKOTA

The interface between DAKOTA and OpenFOAM was created using the basic black box interface provided with the DAKOTA distribution. This interface is a csh script which uses other scripting languages and system calls to handle pre- and post-processing of files for external simulation codes. The csh script calls python scripts to edit input and geometry files and to calculate averages. It calls gmsh to mesh the geometry, and also creates a bash script to run the OpenFOAM simulations on the cluster. Figure 5.1 illustrates how the DAKOTA-OpenFOAM interface is included in the surrogate-based optimization procedure. The sections that follow describe in detail how these steps are carried out.
5.2.1 Optimization Variables

The airfoil optimization focuses on the lift to drag ratio of the inflatable airfoil. The parameters used to optimize the lift to drag ratio are the number of bumps ($n$) along the top and bottom surfaces of the airfoil and the height ($h$) of each bump. The bump height is measured as the distance between the midpoint of the arc representing the bump and the midpoint of a line connecting the endpoints of the arc (fig. 5.3). The leading and trailing edges of the bumpy airfoil are retained and remain unaltered by the optimization.

5.2.2 Sample Points

DAKOTA generates sets of variables for the sample points in the optimization. The csh script which acts as the interface between DAKOTA and other programs invokes a python script which creates an interface between the csh script and Gmsh. This python script reads the values from a DAKOTA parameters file and inserts them into the Gmsh file that will handle the creation of a new airfoil and the computational mesh around it.

5.2.3 Airfoil Geometry

The optimization process needs to automatically generate multiple geometries for the
optimization, but the original bumpy airfoil is not easily modified so a new airfoil based on the original was created in Gmsh. The leading and trailing edges of the original airfoil were retained and used as starting points for the creation of the new airfoil. Tension splines were created using the trough points between the bumps on the top and bottom surfaces of the original airfoil (fig. 5.2). These splines were used as a basis to place the trough points of the new changeable airfoil. Gmsh reads in the user specified number of bumps ($n$) and divides the domain of the spline into $n$ equal segments. The bumps of the original airfoil have a shorter arc length near the leading and trailing edges. In order to match the original airfoil as nearly as possible, the trough points of the new airfoil were clustered near the edges using the following scheme to yield varying arc lengths:

$$y = \frac{L(1 + \tanh(\delta(x/L - 1/2)))}{2\tanh(\delta/2)}$$

where $L$ is the length of the spline, $x$ is the original position along the spline, $\delta$ is the stretching factor, and $y$ is the new position along the spline.

The clustered trough points are then placed along the surface of the airfoil. Using the distance between trough points ($t_1$ and $t_2$ in fig. 5.3) and the user specified value for bump height ($h$), the position of the center point ($c$) is found for a circle that contains the trough points. $\|t_2 - t_1\|/2$, $h$, $r = \|t_1 - c\| = \|t_2 - c\|$, and $r - h$ were used in the Pythagorean theorem to find the radius $r$ of each bump.

The trough points and the center points are then used to create an arc to represent an individual bump. This process is repeated along the top and bottom surfaces of the
airfoil to create all of the bumps. Fig. 5.4(a) and fig. 5.4(b) illustrate the difference between airfoils with and without bump clustering. The clustering displayed is exaggerated for the sake of visualization.

The bump height and the number of bumps acted together in a way that introduced limits on the range of geometries that could be created. If there are too many bumps and the bump height is too high near the leading and trailing edges of the airfoil, the radius of the bumps could be smaller than the bump height. This causes the bumps to invert and create cavities on the surface of the airfoil instead of bumps protruding from it. To prevent this limits were imposed on the number of bumps and on the bump height. The limit on the number of bumps is $7 < n < 22$ while the limit on the bump height is $8.0 \times 10^{-3} m < h < 2.0 \times 10^{-2} m$.

### 5.2.4 Computational Mesh

Once the bumps are in place a 2-d unstructured computational mesh is constructed around the airfoil (fig. 5.6). Using a structured mesh with this particular geometry led to highly skewed cells that did not effectively capture the shape of the bumps. Problems were most evident near the trough points, where grid accuracy is very important for investigating the effects of the bumps on the flow. Refinement helped reduce skewness but at the same time drastically increased the number of cells, which is undesirable when running a large number of individual simulations.

The unstructured mesh could be highly refined near the surface of the mesh and remain
Fig. 5.4: Example of Clustering Applied to Bumpy Airfoil.

(a) Clustered Bumps.

(b) Non-Clustered Bumps.

Fig. 5.5: Possible Airfoils Created During Optimization.

(a) \( n = 8, \ h = 0.02 \) m

(b) \( n = 15, \ h = 0.009 \) m

(c) \( n = 22, \ h = 0.003 \) m
coarse near the boundary while the structured grid could not. The refinement levels were controlled using a characteristic length on the surface of the airfoil, a control box surrounding the airfoil, and the mesh outer boundary. The characteristic length was the distance between two adjacent cell vertices on the surface. It is the smallest on the surface of the airfoil, gets larger on the box around the airfoil, and has the largest value for the outer boundary, thereby creating small cells near the airfoil that grow larger as they get farther away.

The boundaries of the mesh are wedge shaped (fig. 5.6(b)), which allows the number of cells to be reduced as compared to a square boundary to save computational expense without losing accuracy. The leading face of the wedge has a height of 4c, the outlet has a height of 10c, and an overall length of 10c parallel to the chord of the airfoil. The number of cells will vary slightly with the geometry of the airfoil but a typical wedge shaped mesh for the optimization has 100,300 cells. A square shaped mesh of 10c by 10c obtained identical results to the wedge shaped mesh.

The final step in the meshing process was to extrude the 2d mesh to a depth of 1 cell to create a 3d mesh, which is required for OpenFOAM. Extruding the triangular mesh resulted in prism shaped cells. When meshing is completed another python script edits the boundary type definitions which are reset during meshing. The mesh is now ready to be decomposed for parallel running.

5.2.5 Function Evaluations

Once the mesh is decomposed into parallel domains the case is ready for submission. The simulator script that acts as an interface between DAKOTA and OpenFOAM creates a job script for submission to the cluster. Despite its multilevel parallelism capabilities DAKOTA was unable to handle the parallelism of OpenFOAM while running on the Wasatch cluster due to incompatibilities between DAKOTA and the scheduling software. It was necessary to execute DAKOTA in parallel on the login node in order to submit simulations to the PBS queueing system that controls the computational resources. DAKOTA ran with a single level of parallelism but submitting parallel simulations to the queue created a hybrid multilevel approach.
Fig. 5.6: Example of Unstructured Computational Mesh Created for Optimization.
Since DAKOTA had to use PBS to submit jobs, it did not have direct access to monitor each OpenFOAM simulation. To monitor completion of jobs the simulator script captured the job id when each job was submitted to the cluster. Using a while-loop, each job id was periodically compared to the id’s of jobs active on the cluster. If the job id was not active the script checked to make sure the job ran to completion. If the job had completed then the loop would exit and move on to the next step of calculating lift and drag, but if it had not completed the loop would automatically resubmit the job and wait for completion. The loop compared job id’s every 300 seconds, putting the process to sleep between each check to reduce the load on the login node.

Occasionally a simulation would end prematurely and resubmitting the job did not result in a completed simulation. This would cause the optimization to hang because DAKOTA must wait until all simulations for a particular surrogate model iteration are completed before continuing. A simple solution was to change the time step of the simulation, although this would cause the simulation to run slightly longer. Changing the time step had the potential to cause problems on its own since the while-loop checked for a specific time directory while monitoring for completion. Changing the time step could result in a different time directory than expected. The loop would act as if the job had not completed and DAKOTA would hang as it continually resubmits the job. To end this loop the user could simply create the specific time directory without stopping the optimization or affecting the results of the OpenFOAM simulation.

The lift and drag coefficients of the airfoil were used as a guide to measure OpenFOAM convergence and due to unsteadiness in the flow convergence was oscillatory. The oscillations were limited to ten complete cycles and took place over a period of 6,433 time steps. Since the convergence is oscillatory no single value was returned for lift or drag, which is necessary for the optimization to move forward. A python program averaged the lift and drag coefficients over the interval of the steady oscillations and these average values were then used to calculate a lift to drag ratio that was returned to DAKOTA as the objective function value.
For each OpenFOAM simulation a very small initial time step was used to initialize the flow field, resulting in a maximum Courant number of 0.016 after 2000 time steps. The time step was then increased and the simulation was carried out for an additional 17,500 time steps with the final Courant number having a maximum value of 0.67. Each OpenFOAM simulation ran for as little as 4 hours, while occasionally running for up to 5 hours.

5.3 Airfoil Surrogate-Based Optimization with Genetic Algorithm

5.3.1 Surrogate-Based Optimization Setup

A quadratic polynomial data fit surrogate model was used in combination with a genetic algorithm to optimize the airfoil. During initial tests eight sample points were used for each iteration of the surrogate model due to a lack of resources. Poor accuracy led to an increase in the number of samples used for each iteration. After testing was complete additional resources became available for the optimization so the number of sample points for an iteration was increased to 24.

Resources were still limited, however, so only 12 sample points could be evaluated on the cluster simultaneously. DAKOTA generated the sample points and assigned them to individual DAKOTA processes. Each process was assigned two simulations to perform which could only run one after the other. As a result, each iteration of the surrogate model took twice the time possible for evaluations. In terms of computational time the efficiency of the surrogate model is very dependent upon the computational resources available.

The surrogate model went through ten iterations with a soft convergence limit of five specified. The initial trust region covered the entire range of variables with specifications that allow the trust region to contract to 85% of its previous value for poor accuracy or allow the trust region to expand to 125% of its previous value for excellent accuracy.

The genetic algorithm used to optimize the surrogate model used a population size of 10 with a maximum of 15 iterations. There was no crossover with a 100% chance of mutation of sample points. Additional details of the surrogate-based optimization with genetic algorithm setup are contained in the DAKOTA input file included in appendix B.1.
5.3.2 Surrogate-Based Optimization Results

The surrogate-based optimization was performed at three different angles of attack: 2°, 5°, and 10°. Two optimizations were performed for each angle of attack to test whether the same optimum value would be returned for different simulations. Table 5.1 lists the airfoil geometries that were returned from the surrogate-based optimizations at each angle of attack. The only optimization that was able to return an identical airfoil for repeated optimizations was the one for a 10° angle of attack. The two optimizations at a 2° angle of attack had the same initial conditions but did not return the same solution. The resulting airfoils were similar, with a difference of 1 bump and 1.1 mm in the bump height with the lift to drag ratios of the two airfoils being fairly close. The optimization at a 5° angle of attack resulted in greater differences in the number of bumps and the lift to drag ratios while the bump heights only differed by 0.2546 mm.

For each optimization the number of bumps stays very close to the original value, as can be seen when comparing the initial number of bumps ($n_i$) to the final number of bumps ($n_f$) in table 5.1. The final bump height, however, was always lower than the initial value with most of the airfoils returning the same optimum bump height. The lowest bump height returned was the lower limit for the given range of $h$.

Table 5.1: Optimal Airfoils from Surrogate Optimization.

<table>
<thead>
<tr>
<th>$\alpha$ of optimization</th>
<th>$n_f$</th>
<th>$h_f$</th>
<th>$\text{Cl/Cd}$</th>
<th>$n_i$</th>
<th>$h_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>15</td>
<td>9.09375e-3</td>
<td>1.743</td>
<td>15</td>
<td>1.5e-2</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>8.0e-3</td>
<td>2.056</td>
<td>15</td>
<td>1.5e-2</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>8.2546e-3</td>
<td>5.429</td>
<td>11</td>
<td>1.2e-2</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>8.0e-3</td>
<td>7.002</td>
<td>10</td>
<td>1.0e-2</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>8.0e-3</td>
<td>5.521</td>
<td>15</td>
<td>1.5e-2</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>8.0e-3</td>
<td>5.521</td>
<td>15</td>
<td>1.5e-2</td>
</tr>
</tbody>
</table>

The results from all of the OpenFOAM simulations were plotted on a single chart for each angle of attack with the data grouped by bump height (figs. 5.7, 5.9, and 5.11). At each angle of attack the trend in the OpenFOAM simulations was for airfoils with lower bump heights to have an higher lift to drag ratio. The plots also show that small variations in the bump height can produce large variations in the lift to drag ratio. To gain a better
understanding of the data linear trendlines were applied to the results. The trendlines displayed in figures 5.8, 5.10, and 5.12 reveal the typical behavior of the simulations. They better illustrate that a lower bump height leads to a higher angle of attack. The relative lift to drag values of the trendlines also show that reducing the number of bumps can lead, in general, to an increase in the angle of attack. The trendlines tend to have an increasingly higher intercept of the lift to drag axis as the number of bumps decreases.

Fig. 5.7: Cl/Cd Data from Optimization at $\alpha = 2$.

The trendlines tended to be more meaningful for data sets for which there is a large range in bump heights for each bump. Because the surrogate optimization did not always match OpenFOAM results well the trust region contracted towards the lower range of bump heights for most iterations. This resulted in more simulations for airfoils with lower bump heights. Varying the bump height produced large variations in the lift to drag ratio so the trendline slope for sets with a small range in bump height was typically higher than those for which $h$ has a larger range. This can be seen in the trendlines for the optimization at 10° (fig. 5.12). The trendlines for nine bumps and twelve bumps would lead to the conclusion
Fig. 5.8: Trendlines of Cl/Cd from Optimization at $\alpha = 2$.

Fig. 5.9: Cl/Cd Data from Optimization at $\alpha = 5$. 
Fig. 5.10: Trendlines of Cl/Cd from Optimization at $\alpha = 5$.

Fig. 5.11: Cl/Cd Data from Optimization at $\alpha = 10$. 
that twelve bumps would return the best lift to drag value, but the data in figure 5.11 shows that an airfoil with nine bumps will result in the best lift to drag ratio.

5.4 Airfoil Optimization with Genetic Algorithm

The airfoil was also optimized using the coliny_ea algorithm by itself as a basis of comparison for the surrogate model. The genetic algorithm optimization used the same DAKOTA-OpenFOAM interface as the surrogate-based optimization.

The genetic algorithm creates a random sample set and submits the simulations to the cluster. Once the simulations are completed and lift to drag ratios returned, the samples are tested for fitness. The most optimal points are used to propagate new sample points for new simulations. This process continues for the number of iterations specified by the user or until the maximum number of function evaluations is reached.
5.4.1 Genetic Algorithm Setup

The genetic algorithm optimization used a population size of eight with a maximum of ten iterations. Enough resources were available for all eight simulations to take place concurrently. A 100% mutation rate was specified with no specification of mutation scale or mutation range, no crossover was used, and eight new solutions were generated for each iteration of the algorithm. For additional details of the algorithm setup see the input file in appendix B.2.

5.4.2 Genetic Algorithm Optimization Results

The genetic algorithm was used to optimize the airfoil twice at a 5° angle of attack. Both optimizations from the genetic algorithm returned the same airfoil (table 5.2). One of the surrogate-based optimizations at 5° returned the same bump height and lift to drag ratio but a different number of bumps.

Table 5.2: Best Airfoil from Genetic Algorithm Optimization.

<table>
<thead>
<tr>
<th>α of optimization</th>
<th>n_f</th>
<th>h_f</th>
<th>Cl/Cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>8.00e-3</td>
<td>4.288</td>
</tr>
</tbody>
</table>

5.5 Comparison of Optimization Methods

The optimum airfoil returned by the surrogate optimization was always an airfoil predicted by the surrogate model to have the best performance. The predicted lift to drag ratio of the optimum airfoil did not always have a similar lift to drag ratio when tested against an OpenFOAM simulation. Occasionally an OpenFOAM simulation used to create the surrogate models had a better lift to drag ratio than the optimum airfoil returned by the simulation. One instance of the optimizations at 10° returned an optimal airfoil that agreed perfectly with OpenFOAM simulations, while the optimizations at 2° and 5° did not return any. The best airfoils from OpenFOAM simulations are listed in table 5.3.

The surrogate-based optimization was set to run for 10 iterations and construct the surrogate model from the results of 24 simulations during each iteration. However, com-
putational resources were limited and only 12 simulations could run simultaneously. Each DAKOTA process created and submitted two OpenFOAM jobs but DAKOTA could only submit one at a time and the other had to wait, meaning that each iteration of the surrogate model took twice the time it potentially could. The accuracy check of the surrogate model also added time to each iteration. The total compute time for each iteration of the surrogate model was triple the compute time for an individual OpenFOAM simulation.

Performed prior to the availability of more resources, the genetic algorithm was set to use 8 simulations for each of its 10 iterations. Each iteration of the GA could run all of the simulations simultaneously so each iteration of the GA only took the amount of time required to run a single OpenFOAM simulation and one third the time it took to run each iteration of the surrogate model. Table 5.4 lists the number of simulations and the time each optimization method ran at the various angles of attack. Also listed is the computational time saved through the use of surrogate model evaluations. The time savings calculations are based on individual OpenFOAM simulations taking 4 hours with 12 SBO simulations running simultaneously.

Table 5.3: Best Airfoils from OpenFOAM Simulations in Surrogate Optimization.

<table>
<thead>
<tr>
<th>α of optimization</th>
<th>n_f</th>
<th>h_f</th>
<th>Cl/Cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>8.96e-3</td>
<td>2.357</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8.29e-3</td>
<td>1.886</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>9.03e-3</td>
<td>4.1203</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>8.0e-3</td>
<td>4.288</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>8.0e-3</td>
<td>5.521</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>1.02e-2</td>
<td>5.517</td>
</tr>
</tbody>
</table>

Table 5.4: Comparison of Run Times for SBO and GA.

<table>
<thead>
<tr>
<th>method</th>
<th>FOAM sims</th>
<th>surrogate sims</th>
<th>run time (hrs)</th>
<th>time saved (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA at 5°</td>
<td>88</td>
<td>N/A</td>
<td>44</td>
<td>N/A</td>
</tr>
<tr>
<td>SBO at 2°</td>
<td>251</td>
<td>1660</td>
<td>124</td>
<td>553</td>
</tr>
<tr>
<td>SBO at 2°</td>
<td>151</td>
<td>996</td>
<td>84</td>
<td>332</td>
</tr>
<tr>
<td>SBO at 5°</td>
<td>201</td>
<td>448</td>
<td>100</td>
<td>149</td>
</tr>
<tr>
<td>SBO at 5°</td>
<td>176</td>
<td>1162</td>
<td>88</td>
<td>387</td>
</tr>
<tr>
<td>SBO at 10°</td>
<td>176</td>
<td>1162</td>
<td>88</td>
<td>387</td>
</tr>
<tr>
<td>SBO at 10°</td>
<td>148</td>
<td>1162</td>
<td>76</td>
<td>387</td>
</tr>
</tbody>
</table>
The GA took less time to run than any of the surrogate-based optimizations. Part of this is due to the fact that fewer simulations were run by the GA, but also because each iteration of the SBO had to run an accuracy check simulation after the surrogate model was optimized. Had there been enough resources to run all 24 simulations at the same time and had the GA also used 24 sample points for each iteration, the GA would run the same number of iterations in roughly half the time.

The GA could double the population size or the number of iterations and run in the same amount of time as the surrogate optimization. Since more simulations will be used this should increase the accuracy of the GA. But according to the results of optimizations on the Ackley function and Rastrigin’s function (see fig. 4.8), there is no guarantee that such increases would improve the accuracy of the GA.

Figure 5.13 shows the progression of the solutions for the GA and some of the SBO’s for each angle of attack. As stated previously the GA ran fewer overall simulations and took less time than any of the SBO’s while following the trends predicted by the SBO’s. Based on this fact alone it would seem that the GA is better suited to this problem. However, comparing the progression of the solutions in figure 5.13, one can see that the surrogate models all converged to their optimal lift to drag ratio on the third iteration while the GA did not find its optimal airfoil until the fifth iteration. Furthermore, the GA does not converge around the optimal geometry or lift-to-drag ratio like the SBO did.

The SBO tests the accuracy of its model and can terminate once sufficient accuracy has been met, avoiding unnecessary use of computational resources. The SBO could be set to terminate once two consecutive iterations returned the same optimum, decreasing the overall time and number of simulations. The GA has no such capability. The only way to check whether the GA has the best solution is to run more simulations and take up resources. By using more computational resources and lowering the convergence limit the SBO might have been able to find an optimal value in less time than the GA did while giving more certainty as to the accuracy of the solution than the GA can.
Fig. 5.13: Comparison of Lift-to-Drag Ratio vs Number of FOAM Simulations.
Chapter 6
Conclusions and Discussion

6.1 Surrogate Model With Genetic Algorithm

Although the genetic algorithm took less overall time to come to a solution and the results followed the trends predicted by the surrogate model, the surrogate-based optimization performed better. As discussed in the next section, the surrogate-based optimization with genetic algorithm returned better airfoils. This was done while saving up to 553 hours of computing time on 192 processors. The surrogate model was also able to reduce the effects of noise created by changing the geometry of the airfoils.

6.2 Airfoil Performance

The optimized airfoils for $2^\circ$ and $10^\circ$ were swept through a range of angles of attack from $0^\circ$ to $20^\circ$ using the first order discretization scheme and then the second order discretization scheme. The smooth, original bumpy, and modified bumpy airfoil (starting geometry in optimizations) were also swept through the same angles of attack for comparison.

The results for angles of attack larger than $7.5^\circ$ were omitted because the lift and drag coefficients did not display steady oscillations at $12.5^\circ$ and up for the first order scheme and at $10^\circ$ and up for the second order scheme, hence averages taken would not be a good measurement of lift to drag ratios.

Figures 6.1, 6.2, and 6.3 compare the first and second order results. The first order results follow very well from the results of the optimization. The lift to drag ratios and lift coefficients for the surrogate optimized airfoils are higher than those for the original bumpy airfoil but not as high as those for the smooth airfoil. The airfoil from the $2^\circ$ optimization has the best values. The drag coefficients for the surrogate optimized airfoils are lower than the drag for the bumpy airfoil and some of the drag coefficients for the smooth airfoil.
The second order results are slightly different from the first order results. The overall lift to drag ratios tended to be lower for second order schemes than first order. For the second order scheme the optimized airfoils have lift to drag ratios that are very close to the lift to drag ratio of the original bumpy airfoil, but the optimized airfoils no longer show the same improvement that the first order airfoils show. However, the values only changed slightly between first and second order so an optimization carried out with a second order discretization scheme should still follow the same trends as the first order scheme.

There are slight differences in the geometries of the original bumpy and modified bumpy airfoils, which created differences in performance. Only comparing the optimized airfoils to the original bumpy airfoil is not the best measure of performance for the optimization. Although second order results do not show an improvement over the original bumpy airfoil, the surrogate optimizations show an improvement over the modified bumpy airfoil used as a starting point in all optimizations.

The airfoil optimized by the genetic algorithm, although it followed the trends suggested by the optimization process, did not produce an improvement over the original bumpy airfoil for first or second order results, which was a surprising result.

Out of all of the airfoils, the best overall came from the surrogate-based optimizations. The two with the best performance are the airfoils optimized with the surrogate models at 2° and 10° angles of attack. They both outperformed the airfoil optimized with the genetic algorithm at 5° and the surrogate optimization at 5°. The 2° and 10° airfoils are very close in performance but for angles of attack of 5° and 7.5° the airfoil optimized at 10° shows the best performance with second order discretization.

6.3 Flow Separation

One goal in maximizing the performance of the airfoil was to reduce or eliminate the size of the separation bubble forming on the upper surface of the airfoil. The velocity profiles of the airfoils optimized at 2° and 10° were compared visually to the velocity profile of the original bumpy airfoil at a time when the airfoils have a similar trailing vortex. The 2° optimized airfoil was compared to the original airfoil at a 2° angle of attack (fig. 6.4). The
(a) 1st Order.  
(b) 2nd Order.

Fig. 6.1: Plots of $C_l/C_d$ vs $\alpha$ for 1st and 2nd Order Results.

(a) 1st Order.  
(b) 2nd Order.

Fig. 6.2: Plots of $C_l$ vs $\alpha$ for 1st and 2nd Order Results.

(a) 1st Order.  
(b) 2nd Order.

Fig. 6.3: Plots of $C_d$ vs $\alpha$ for 1st and 2nd Order Results.
10° optimized airfoil was compared to the original airfoil with both at a 5° angle of attack. For both comparisons the optimized airfoils showed no noticeable change in the separation bubble.

![Velocity Profiles for the Original and 2° Optimized Airfoils at α = 2.](image)

Fig. 6.4: Velocity Profiles for the Original and 2° Optimized Airfoils at α = 2.

![Velocity Profiles for the Original and 10° Optimized Airfoils at α = 5.](image)

Fig. 6.5: Velocity Profiles for the Original and 10° Optimized airfoils at α = 5.

### 6.4 Summary and Future Work

A bumpy airfoil based on the Eppler 398 profile was optimized using a genetic algorithm and a surrogate-based optimization. The surrogate-based optimizations saved up to 553 hours of simulation time using the computational resources at USU. The best airfoil found during the course of the optimizations was from a surrogate-based optimization at 10° and had 15 bumps with a bump height of 8 mm, which is the same number of bumps as the original bumpy airfoil but smaller radii.

The surrogate-based optimizations had limited accuracy due to ”noise,” or variations in the lift to drag ratios as a result of small variations in bump height. The changes in airfoil geometry also change the computational mesh which can change the discretization error. This in turn can create additional noise in the simulations. The surrogate model can
suppress noise in the function values and allow extraction of trends in the data. The trend of the optimizations was that reducing the bump height or the number of bumps would increase the lift to drag ratio of the airfoil. Based on this trend one would expect that the best airfoil from an improved optimization would be one with the lowest number of bumps and bump height possible.

Using more sample points to create the surrogate model for each iteration would result in a more accurate model. This would result in a more accurate overall solution. Using a second order scheme should also increase the accuracy of each OpenFOAM solution and perhaps reduce the amount of noise present in the OpenFOAM results, which would be another benefit for creating a more accurate surrogate model.

The modifiable airfoil created for the optimizations had a constant bump height for all bumps along the surface of the airfoil. The original bumpy airfoil had a constant radius for the bumps and since they had different arc lengths the bump height was slightly different for each of the bumps with the largest bump near the 1/2 chord of the airfoil. The constant bump height was used because it was difficult to change the radius to control the shape of the airfoil without creating undesirable effects. Changing the radius also offered less direct control over the bumps, making it more difficult to create small changes in the height of the bumps. It might be beneficial to find a way to use a constant radius across the airfoil surface, perhaps by controlling the height of one bump and then using the radius of that bump for the rest of the bumps. This should offer more direct control over the bumps than just changing the radius alone while producing varying bump heights.

The bumps help create lateral strength in a wing constructed from the 2D airfoil section. Changing the number of bumps or the bump height may affect the weight the wings can support, possibly resulting in an airfoil that will perform well aerodynamically but not structurally. Using finite element analysis to test the strength of an inflatable wing as part of the optimization might impose lower limits on the number and size of bumps allowing for the creation of the best possible inflatable wing.
References


Appendices
Appendix A

DAKOTA Ackley Function Test Input Files

A.1 DAKOTA input file for Ackley function surrogate-based optimization.
# DAKOTA INPUT FILE - dakota_sbo_ackley
# Surrogate-based optimization to minimize Ackley's function.
# This file is formatted for use with DAKOTA version 4.1

strategy,
    surrogate_based_opt
    tabular_graphics_data
    max_iterations = 100
    soft_convergence_limit = 10
    opt_method_pointer = 'COLINY_EA'

# the trust region (TR) commands specify the
# size of the first trust region, plus the
# scaling factors that are applied to the TR
# on subsequent iterations

trust_region
    initial_size = 0.3
    minimum_size = 1.0e-6
    contract_threshold = 0.25
    expand_threshold   = 0.75
    contraction_factor = 0.75
    expansion_factor   = 1.25

# begin opt specification

method,
    id_method = 'COLINY_EA'
    model_pointer = 'SURROGATE'
    coliny_ea
        population_size = 100
        max_iterations = 1000
        max_function_evaluations = 100000
        crossover_rate 0.0
        mutation_rate 1.0
        mutation_scale = 0.1
        mutation_range = 2
        fitness_type linear_rank
        crossover_type uniform
        replacement_type random = 0
        new_solutions_generated = 10
        initialization_type unique_random
        mutation_type replace_uniform

model,
id_model = 'SURROGATE'
surrogate global
  responses_pointer = 'SURROGATE_RESP'
dace_method_pointer = 'SAMPLING'
### Section to specify surface fit method.
  polynomial quadratic

variables,
  continuous_design = 2
    cdv_initial_point  0.1  0.1
    cdv_lower_bounds   -5.0  -5.0
    cdv_upper_bounds   5.0   5.0
    cdv_descriptor     'x1'  'x2'

responses,
  id_responses = 'SURROGATE_RESP'
  num_objective_functions = 1
  numerical_gradients
    method_source dakota
    interval_type central
    fd_gradient_step_size = 1.e-6
  no_hessians

# Sampling method specifications for sampling in
# the trust regions of the SBO strategy
method,
  id_method = 'SAMPLING'
  model_pointer = 'TRUTH'
  nond_sampling
    samples = 20
    seed = 531
    sample_type lhs
    all_variables

model,
  id_model = 'TRUTH'
  single
    responses_pointer = 'TRUE_RESP'

interface,
  analysis_driver = 'ackley_simulator_script'
  parameters_file = 'params.in'
  results_file = 'results.out'
  file_tag file_save aprepro
A.2 DAKOTA input file for Ackley function genetic algorithm.
# DAKOTA INPUT FILE - dakota_ga_ackley.in
# This input file optimizes the Ackley function using
# the coliny_ea genetic algorithm

strategy,
  single_method
  tabular_graphics_data

method,
  coliny_ea
    population_size = 5
    max_iterations = 10
    max_function_evaluations = 400
    crossover_rate 0.0
    mutation_rate 1.0
    mutation_scale = 0.1
    mutation_range = 2.0
    fitness_type merit_function
    crossover_type uniform
    replacement_type random = 0
    new_solutions_generated = 10
    initialization_type unique_random
    mutation_type replace_uniform

variables,
  continuous_design = 2
    cdv_initial_point  0.1  0.1
    cdv_lower_bounds  -5.0  -5.0
    cdv_upper_bounds   5.0   5.0
    cdv_descriptor    'x1'  'x2'

interface,
  system
    analysis_driver = 'ackley_simulator_script'
    parameters_file = 'params.in'
    results_file    = 'results.out'
    file_tag file_save aprepro

responses,
  num_objective_functions = 1
  numerical_gradients
    fd_gradient_step_size = .0000001
  no_hessians
Appendix B

DAKOTA Airfoil Optimization Input Files

B.1 DAKOTA input file for airfoil surrogate-based optimization.
# DAKOTA INPUT FILE - dakota_sbo_bumpy.in

# Surrogate-based optimization to minimize a bumpy airfoil.  
# For use with DAKOTA 4.2

strategy,
    single_method
    method_pointer = 'SBLO'

method,
    id_method = 'SBLO'
    surrogate_based_local
    model_pointer = 'SURROGATE'
    approx_method_pointer = 'COLINY_EA'
    max_iterations = 10
    soft_convergence_limit = 5

# the trust region (TR) commands specify the
# size of the first trust region, plus the
# scaling factors that are applied to the TR
# on subsequent interations

trust_region
    initial_size = 1.0
    minimum_size = 1.0e-6
    contract_threshold = 0.30
    expand_threshold  = 0.85
    contraction_factor = 0.85
    expansion_factor   = 1.25

# begin opt specification

method,
    id_method = 'COLINY_EA'
    coliny_ea
        population_size = 15
        max_iterations = 10
        max_function_evaluations = 150
        seed = 11011111
        crossover_rate = 0.0
        mutation_rate = 1.0
        fitness_type linear_rank
        crossover_type uniform
        replacement_type random = 0
        new_solutions_generated = 50
initialization_type simple_random
mutation_type replace_uniform

model,
  id_model = 'SURROGATE'
surrogate global
  responses_pointer = 'SURROGATE_RESP'
dace_method_pointer = 'SAMPLING'
correction additive zeroth_order

### Section to specify surface fit method.
  polynomial quadratic

variables,
  discrete_design = 1
  ddv_initial_point 15
  ddv_upper_bounds 22
  ddv_lower_bounds 5
  ddv_descriptor 'n'
  continuous_design = 1
  initial_point 0.015
  lower_bounds 0.008
  upper_bounds 0.02
  descriptors 'h'

responses,
  id_responses = 'SURROGATE_RESP'
  num_objective_functions = 1
  numerical_gradients
    method_source dakota
    interval_type central
    fd_gradient_step_size = 1.e-6
  no_hessians

###############################################
# Sampling method specifications for sampling in
# the trust regions of the SBO strategy
###############################################
method,
  id_method = 'SAMPLING'
  model_pointer = 'TRUTH'
nond_sampling
  samples = 24
  seed = 531
  sample_type lhs
  all_variables
B.2 DAKOTA input file for airfoil genetic algorithm.
# DAKOTA INPUT FILE - dakota_ga_bumpy.in
# This Dakota input file optimizes the bumpy airfoil using coliny_ea strategy,
strategy,
  single_method
  tabular_graphics_data
method,
  coliny_ea
    population_size = 8
    max_iterations = 10
    max_function_evaluations = 200
    seed = 1234
    crossover_rate 0.0
    mutation_rate 1.0
    #mutation_scale = 0.1
    #mutation_range = 2.0
    fitness_type linear_rank
    crossover_type uniform
    replacement_type random = 0
    new_solutions_generated = 8
    initialization_type simple_random
    mutation_type replace_uniform
variables,
  discrete_design = 1
    ddv_initial_point  18
    ddv_upper_bounds  22
    ddv_lower_bounds  5
    ddv_descriptor 'n'
  continuous_design = 1
    initial_point  0.011
    lower_bounds  0.008
    upper_bounds  0.02
    descriptor 'h'
interface,
  system
    evaluation_static_scheduling
    analysis_driver = './bumpy_simulator_script'
    parameters_file = 'params.in'
    results_file = 'results.out'
    file_tag file_save aprepro
responses,
  num_objective_functions = 1
  no_gradients
  no_hessians