

Multiple objective optimization of an airfoil shape

by

Antoine Dymond

A dissertation submitted in partial fulfillment
of the requirements for the degree

Master of Engineering

in the

Department of Mechanical and Aeronautical Engineering
Faculty of Engineering, the Built Environment and Information
Technology

University of Pretoria
Pretoria

February 15, 2011

Abstract

An airfoil shape optimization problem with conflicting objectives is handled using two different multi-objective approaches. These are an *a priori* scalarization approach where the conflicting objectives are assigned weights and summed together to form a single objective, and the Pareto-optimal multi-objective approach.

The optimization formulations for both approaches contain challenging numerical characteristics which include noise, multi-modality and undefined regions. Gradient-, surrogate- and population-based single objective optimization methods are applied to the *a priori* formulations. The gradient methods are modified to improve their performance on noisy problems as well as to handle undefined regions in the design space. The modifications are successful but the modified methods are outperformed by the surrogate methods and population based methods.

Population-based techniques are used for the Pareto-optimal multi-objective approach. Two established optimization algorithms and two custom algorithms are implemented. The custom algorithms use fitted unrotated hyper ellipses and linear aggregating functions to search the design space for non-dominated designs. Various multi-objective formulations are posed to investigate different aspects of the airfoil design problem. The non-dominated designs found by the Pareto-optimal multi-objective optimization algorithms are then presented.

Acknowledgements

The following parties are thanked and acknowledged for their contribution to this work:

- Schalk Kok the supervisor,
- Bennie Broughton the co-supervisor, and
- the CSIR for their funding.

CONTENTS

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Problem description | 5 |
| 2.1 | A priori scalarization single objective formulations | 6 |
| 2.2 | Problem characteristics | 8 |
| 3 | A priori scalarization optimization using gradient-based methods | 12 |
| 3.1 | Finite differences | 13 |
| 3.1.1 | Gradient of the objective function | 13 |
| 3.1.2 | Hessian approximation | 17 |
| 3.1.3 | Testing | 18 |
| 3.1.4 | Analysis of the single objective cost function's gradient | 21 |
| 3.2 | Line Searches | 24 |
| 3.2.1 | Golden section search | 25 |
| 3.2.2 | Section populating line search | 26 |
| 3.2.3 | Testing | 29 |
| 3.3 | Unconstrained optimization methods | 31 |
| 3.3.1 | BFGS | 32 |
| 3.3.2 | CGPR | 32 |
| 3.3.3 | LFOP | 33 |
| 3.3.4 | Testing | 35 |
| 3.4 | Constrained gradient-based methods | 37 |
| 3.4.1 | LFOPC | 38 |
| 3.4.2 | Lagrange multiplier method | 39 |
| 3.4.3 | SQP | 41 |
| 3.4.4 | COBYLA | 43 |

| | | |
|--|---|------------|
| 3.4.5 | SLSQP | 44 |
| 3.4.6 | Testing | 44 |
| 3.5 | Airfoil optimization | 46 |
| 4 | A priori scalarization optimization using Population-based methods | 56 |
| 4.1 | Rules governing point selection | 56 |
| 4.2 | Differential Evolution | 57 |
| 4.3 | Particle Swarm | 59 |
| 4.4 | Testing | 60 |
| 4.5 | Airfoil optimization | 62 |
| 5 | Pareto-optimal multi-objective optimization | 70 |
| 5.1 | Definitions | 71 |
| 5.2 | MOPSO | 72 |
| 5.3 | MOSADE | 76 |
| 5.4 | EPO | 79 |
| 5.5 | Testing | 88 |
| 5.6 | Airfoil optimization | 96 |
| 6 | Conclusion | 103 |
| Appendix A - Further discussion on line search algorithm testing results | | 109 |
| Appendix B - Constrained optimization single objective test functions | | 112 |
| Appendix C - Performance of gradient-based algorithms on the single objective constrained test problems | | 127 |
| Appendix D - Performance of the population-based algorithms on the single objective constrained test problems | | 137 |
| Appendix E - Optimization results for first airfoil multi-objective formulation | | 142 |
| Appendix F - Optimization results for the second airfoil multi-objective formulation | | 146 |
| Appendix G - Optimization results for the third airfoil multi-objective formulation | | 151 |

CHAPTER 1

INTRODUCTION

Unmanned Aerial Vehicles, UAVs, have different flight requirements depending on their application. The aim of the UAV's design team is to determine the best design for their application, given time and monetary constraints. Computer-based simulation and optimization routines greatly aid this process. In this work, various optimization routines are implemented, tested and then applied to a subsonic UAV airfoil drag minimization problem.

The airfoil shape optimization has multiple conflicting objectives which are handled using two different approaches. The *a priori* approach reduces the problem to a single objective form by assigning weights to each objective and summing them together. The *a priori* approach is popular amongst users who are familiar with single objective optimization as it allows them to tackle multi-objective optimization problems using single objective methods. The other approach implemented is Pareto-optimal multi-objective optimization.

The primary focus of this work is the optimization algorithms and not the aerodynamic modeling. The problem's modeling has been done by the CSIR [6, 7, 3], with the aerodynamic analysis performed by well-tested software packages. The objective of this dissertation is to replicate and extend the results produced by the CSIR. These extensions are primarily implementing Pareto-optimal multi-objective techniques, and implementing techniques capable of handling the maximum lift constraint. The maximum lift constraint sets the minimum value for the maximum lift coefficient that can be generated by the airfoil, and proved particularly problematic in earlier work at the CSIR.

Practical optimization problems often exhibit challenging characteristics. These challenging characteristics include noise, multi-modality, high computational cost and undefined regions, all of which are present in this application. Two approaches can be used when tackling badly behaved functions; the first is to eliminate the bad behavior, and the

second is to apply robust optimization techniques. Eliminating the bad behavior requires the simulation software be re-written and re-compiled which is not always possible. In this work, the simulation software is treated as a “black-box” with the focus on implementing robust algorithms. Chapter 2 describes the optimization problem’s formulation and characteristics in further depth.

Chapter 3 discusses the single objective gradient-based techniques implemented and the results they obtained for the *a priori* airfoil optimization formulations. The optimization problem does not meet the requirements for which gradients method are developed, but gradient methods can still be employed to improve the function value. Certain parts of the gradient methods are customized in order to improve their performance on the optimization problems. This chapter also includes the results from two surrogate based methods.

Chapter 4 presents the single objective population-based methods implemented and their results for *a priori* optimization formulations. These stochastic methods are more expensive than the gradient methods but are also more robust. The methods are investigated as an alternative to the gradient-based algorithms, as they are developed to handle optimization problems with challenging characteristics such as those exhibited in this drag minimization problem.

The Pareto-optimal approach for multiple objective optimization of the 2D UAV airfoil is documented in Chapter 5. One of the difficulties of the *a priori* approach is choosing the objective weights [20]. Using Pareto-optimal multi-objective optimization eliminates this problem by allowing the practitioner to select designs *a posteriori* [35] from the non-dominated solutions found.

The algorithms are documented by presenting the theory relevant for both understanding and application, followed by the testing of that algorithm’s code. The testing is important as it ensures that the algorithms, most of which have been programmed by the author, are working correctly and that their results can be trusted.

CHAPTER 2

PROBLEM DESCRIPTION

The problem’s modeling involves performing an aerodynamic analysis on a generated airfoil shape. Optimization algorithms use the model in order to determine the optimal design variables, in terms of given objective function/s and constraints. The success of the optimizer depends on how well suited the algorithm and its chosen parameters are for the model’s characteristics. The model and its behavior is discussed in this chapter.

An aircraft’s wing redirects the air that it passes through, generating both a lift force and a drag force. These forces are functions of the wing geometry, and the flight conditions. One of the functions of an aircraft’s wing is generating lift to counteract the gravitational force applied to it. When an aircraft flies at a different angle of attack, different drag and lift forces are generated.

The software package, PROFOIL [30, 29], specializes in airfoil generation and makes use of inverse methods. Direct airfoil methods generate the airfoil’s geometry first, and then computes the velocity distribution around the airfoil. In contrast, the inverse methods transform a velocity distribution into an airfoil. PROFOIL makes use of conformal mapping to achieve this. The direct and inverse methods are illustrated in Figure 2.1.

XFOIL[15, 16] is another open-source aerodynamics program, that amongst other

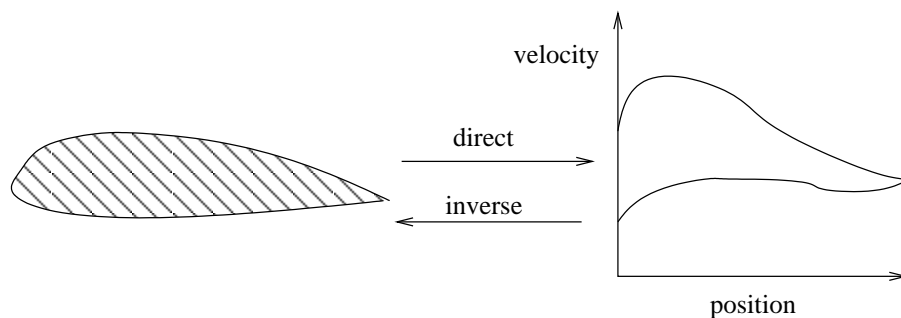


Figure 2.1: Direct and inverse methods for airfoil generation

| Design variable(s) | Description |
|---------------------------|--|
| x_1 | distribution of control-points along top of airfoil |
| x_2 | distribution of control-points along bottom of airfoil |
| x_3, x_4, \dots, x_{14} | velocity at control-points |
| x_{15} | finite trailing edge angle |
| x_{16} | trailing edge recovery parameter |
| x_{17} | thickness ratio |

Table 2.1: Description of the optimization design variables. For further information on PROFOIL’s parameters refer to [28].

features, allows the viscous analysis of subsonic isolated airfoils. XFOIL is used to determine the profile drag created by an airfoil for the angle of attack required to generate a specified lift coefficient.

PROFOIL together with XFOIL are used for the aerodynamic analysis. The design-variables are passed into PROFOIL which generates the shape of the 2-D airfoil. The optimization design variables which are used by PROFOIL to generate the airfoil are listed in Table 2.1. XFOIL then determines the drag coefficients (C_D) for specified lift coefficients (C_L) and Reynolds numbers. The drag coefficient is solved for a constant $Re\sqrt{C_L}$ which is set to 375 000 and for a transition criterion amplification factor of e^9 .

The two different single objective optimization formulations are discussed in the next section, after which the numerical nature of the model is described. The multi-objective formulations are discussed in Chapter 5 and not in this section, as they are based upon the multi-objective context which is only introduced in Chapter 5.

2.1 A priori scalarization single objective formulations

The *a priori* cost function, f , which is a single objective aims to minimize a drag bucket for the required flight envelope. Three drag coefficients C_{D_1} , C_{D_2} and C_{D_3} are used to approximate the drag bucket. C_{D_1} corresponds to the drag coefficient generated for creating the lift required for the UAV in cruise condition. C_{D_2} and C_{D_3} are the drag coefficients for loiter and high speed dash lift requirements. f is a function of the 17 design variables $\mathbf{x} \in \mathfrak{R}^{17}$, and is constructed by blending the three drag coefficient values. For the *a priori* approach C_{D_1} is given higher importance than C_{D_2} and C_{D_3} as the UAV is likely to spend the majority of its flight time in a cruise mode. The *a priori* single objective function is defined as

$$f(\mathbf{x}) = 3C_{D_1}(\mathbf{x}) + C_{D_2}(\mathbf{x}) + C_{D_3}(\mathbf{x}). \quad (2.1)$$

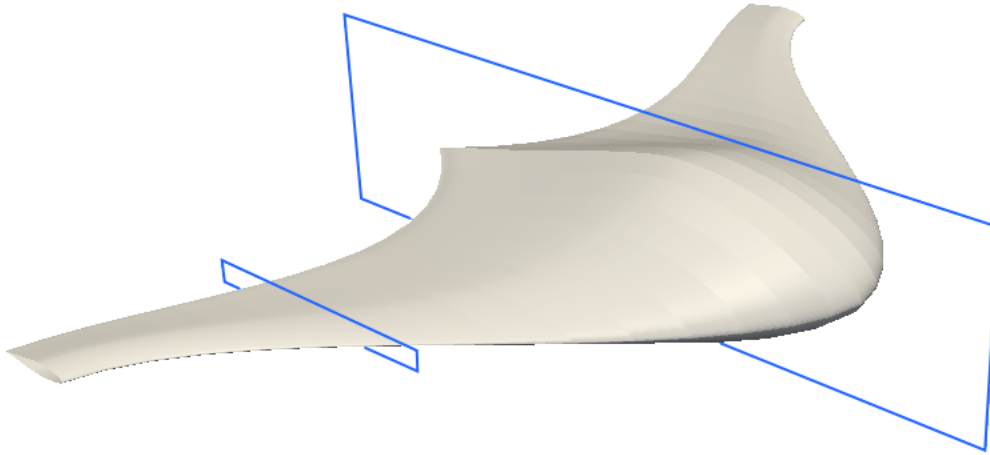


Figure 2.2: The 2D sections of a blended wing-body UAV which can be optimized using the avionics box and maximum lift single objective formulations

The two *a priori* formulations focus on different 2D sections of a blended wing-body UAV. The avionics box height formulation aims to minimize f such that an unrotated box can fit inside the airfoil. This 2D section will be used for the 50 % chord section of the UAV which houses the control box. The maximum lift formulation aims to minimize f so that a maximum lift constraint is met, generating a 2D profile that can be used on the wing tip. Figure 2.2 shows the 50 % chord and wing root sections on a fictitious flying wing UAV.

The avionics box constraint is calculated by determining the maximum height of an unrotated box placed inside the airfoil. This constraint only fails when the airfoil shape cannot be generated by PROFOIL. The height constraint does not contain noise like the objective function (demonstrated in Section 2.2) and is considered “well behaved”.

The maximum lift of an airfoil cannot be directly determined from the model. The work-around implemented is that the maximum lift constraint first checks if the airfoil can generate the required lift. If the airfoil can generate the required lift a violation of zero is returned. If the required lift could not be generated, the minimum solver residual is used as the violation. The solver residual gives a rough estimate of the maximum lift constraint violation, as shown in Figure 2.3.

The maximum lift work-around is noisy and discontinuous which increases the difficulty of the optimization problem. Additional work on the analysis side should reduce these unfavourable characteristics. However this is not done as this goes against the “black-box” approach which is followed for this work.

The maximum lift formulation is more challenging than the avionics box height formulation due to the demonstrated characteristics as well as the fact that it steers the optimizer towards undefined region of the design space. Besides the difficulties associated with undefined regions, the model’s nature, in which multiple solver settings are tried until a solution is determined, also increases the computational expense of each

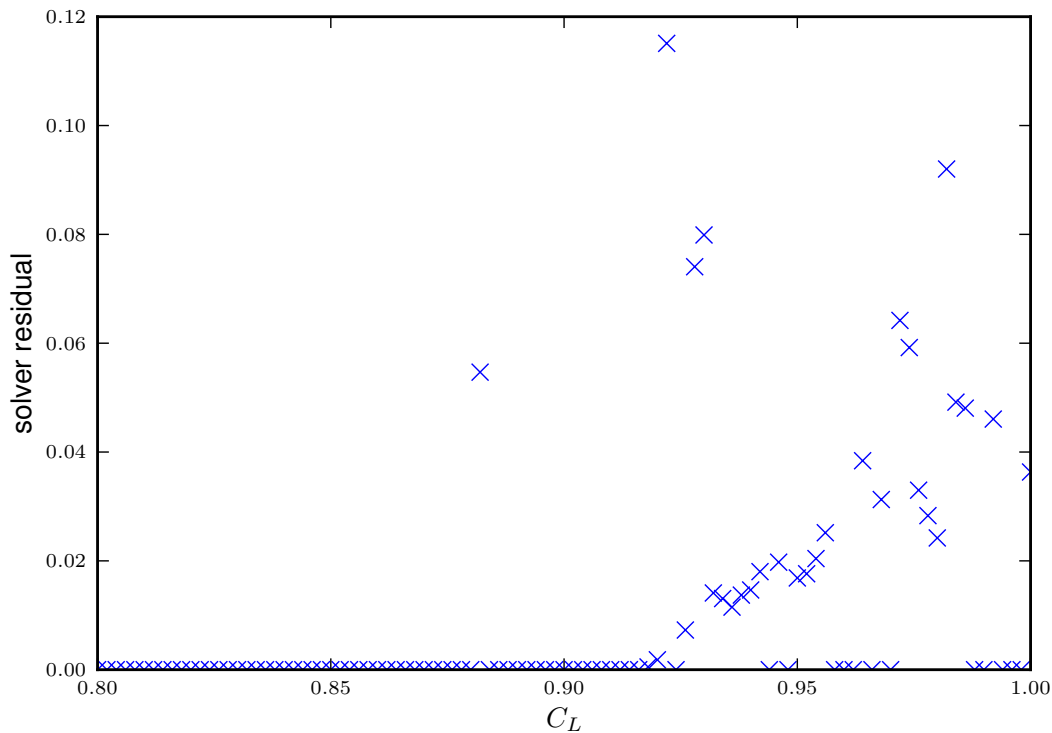


Figure 2.3: The minimum residuals from the XFOIL solver for different coefficients of lift

function evaluation. The numerical behavior exhibited by the model is discussed further in the next section.

2.2 Problem characteristics

In optimization with expensive cost functions, the aim is to determine the optimum in the least amount of time. In most cases where function evaluation time is design-independent, this translates to the least number of function evaluations. Successful methods use characteristics of the function’s behavior in order to enhance performance. If a method is applied to a problem in which the method’s assumptions are false, the method will probably perform poorly. Inspection of an optimization problem’s characteristics grants insight on which methods are expected to perform well. The drag functions exhibit the following behavior:

- undefined sections in the design space
- multi-modal
- noisy
- expensive evaluations

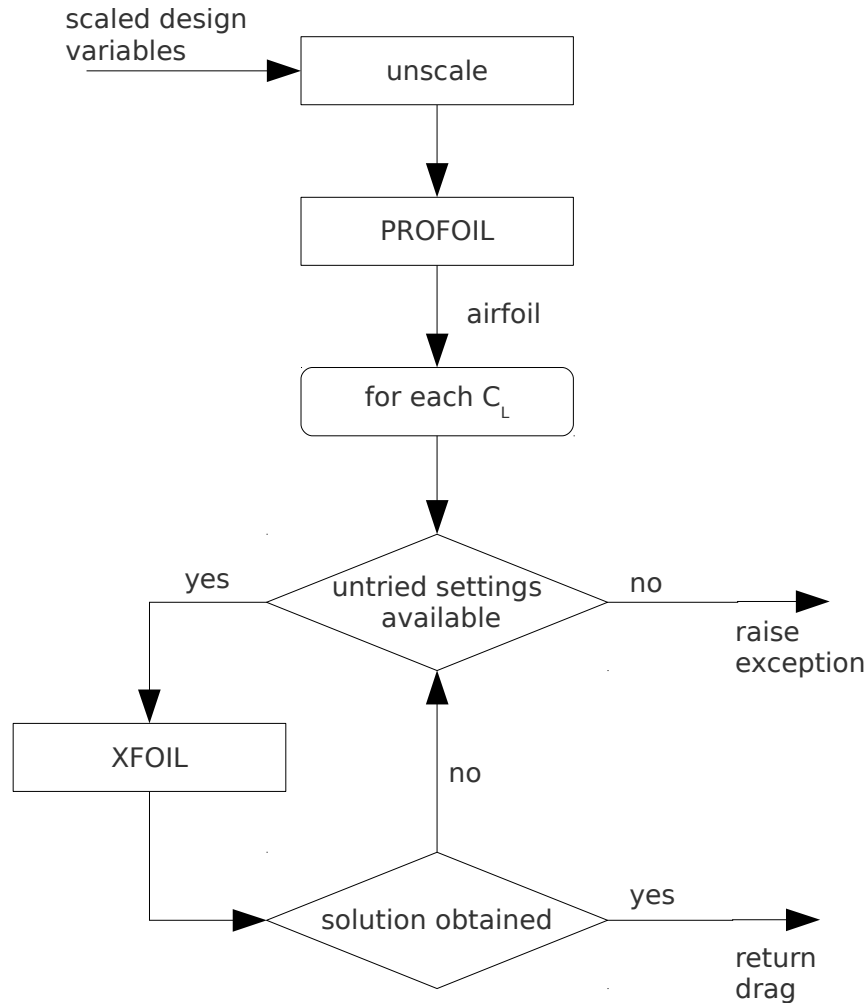


Figure 2.4: Objective function evaluation

- no direct gradient information

The model's characteristics suggest that stochastic-based methods are better suited than gradient-based methods. Gradient methods are proven to work when the function is infinitely differentiable, convex, uni-modal and defined throughout the design space. When these prerequisites are not met, gradient methods are not guaranteed to converge to a minimum or to even perform well. The stochastic population-based methods do not assume smoothness/infinite differentiability, and as such are expected to perform better.

Examining the airfoil modeling reveals where most of the problem behavior originates. The objective function first unscales the design variables, as the design variables have been scaled so that they are all approximately in the domain of 0 to 1. Then PROFOIL attempts to model an airfoil to meet the condition specified by those design variables. Upon success, XFOIL is used to determine the drag coefficients for the specified lift coefficients. A diagrammatic representation of the inner working of the objective function is shown in Figure 2.4.

XFOIL often requires different solver settings to obtain a valid solution. The settings vary the number of steps, step direction, and the viscous acceleration parameter used by XFOIL's Newton solver. The viscous acceleration parameter is used by XFOIL's modified Newton-solver to eliminate smaller entries in the tangent matrix which decreases the computational cost of each iteration [16]. The undefined sections in the design space originate when all the drag coefficients cannot be calculated. The drag analysis also contains high-frequency noise which is artificial and not normally present in XFOIL. A line section through the design space, shown in Figure 2.5, demonstrates these properties.

The function evaluation is undefined when either the airfoil physically cannot achieve the required flight conditions, or the solution exists but the solvers have failed. The solvers rarely fail because the airfoil could not be generated, but mainly because the drag could not be calculated.

Part of the noise on the objective function is artificial and originates from an improperly compiled version of XFOIL and not due the physical nature of the problem. This noise component is deliberate and not removed as it is often present in practical optimization problems.

The computational cost of the function evaluation varies largely. In the majority of cases, XFOIL solves the drag with the first solver settings tried. For more complicated airfoils, multiple XFOIL settings need to be tried before a solution is obtained. When no solution is found after all the settings are tried an exception is raised and passed to the optimizer. The ratio of quickest time to the longest time required for a function evaluation is about 500. A quick evaluation takes about half a second on a single CPU of a T7200 2.00 GHz Intel® Core™ 2 Processor [1].

The next chapter gives details on the gradient-based single objective optimization of the airfoil.

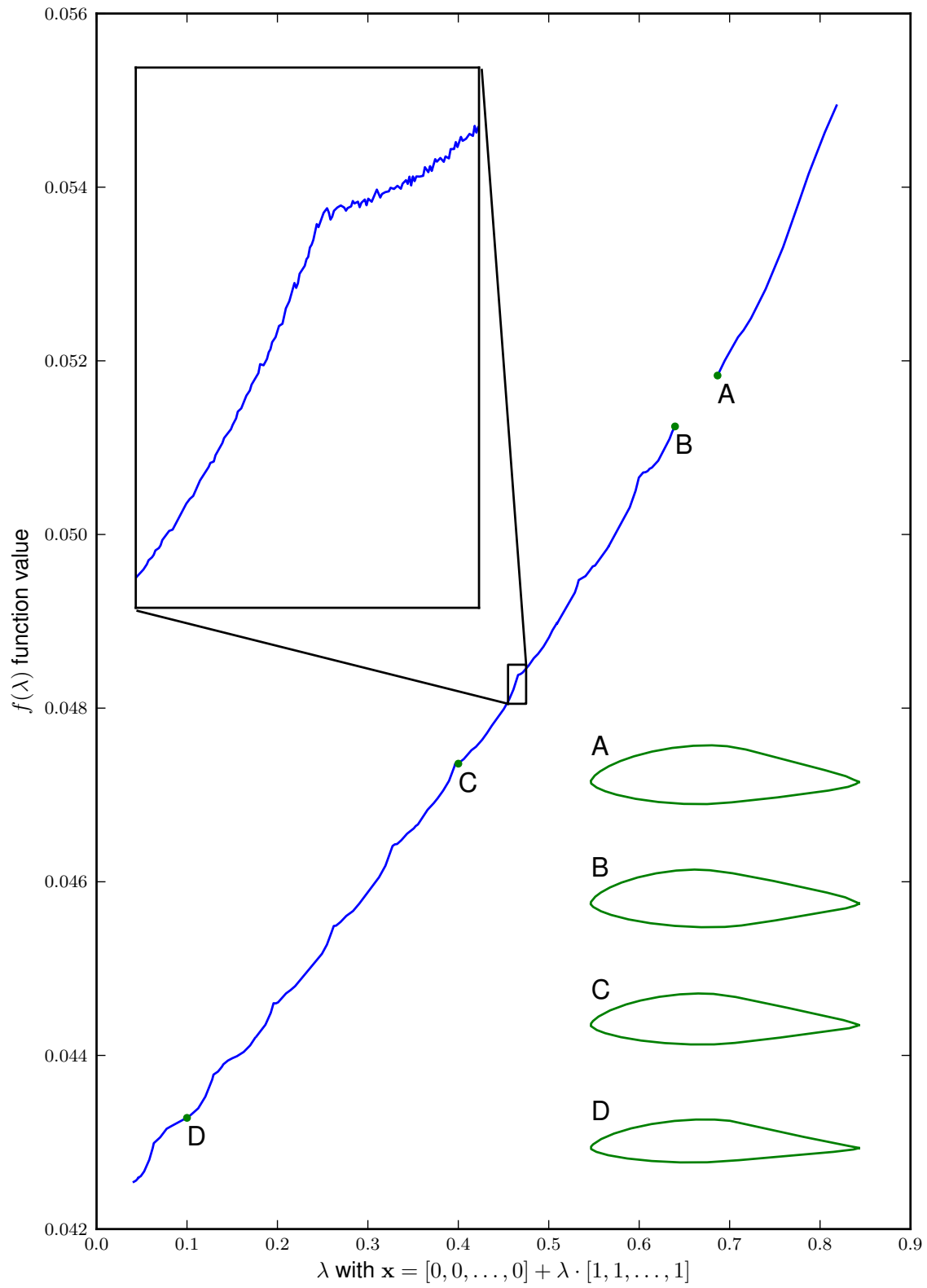


Figure 2.5: The *a priori* approach function value for a line section through the design space.

CHAPTER 3

A PRIORI SCALARIZATION OPTIMIZATION USING GRADIENT-BASED METHODS

Different gradient-based techniques are implemented and applied to the airfoil single objective *a priori* optimization formulations. In addition to the gradient methods two surrogate methods are also used. The techniques implemented and the results achieved are presented in this chapter.

The chosen airfoil optimization formulations do not meet the conditions required for gradient-based optimization. Large sections of the design space are undefined, the problem is multi-modal and the gradient is discontinuous and not smooth. However gradient methods can still be employed to improve the function value, but are not guaranteed to converge to a local minimum.

Gradient methods begin their search of the design space from a starting point, \mathbf{x}_0 . The design space \mathbf{x} consists of n dimensions each in the real set, $\mathbf{x} \in \mathfrak{R}^n$. Gradient information is then used to improve the function value and to attempt to determine the objective function minimum. The objective function's gradient f' , expressed as a column vector is:

$$f' = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad \mathbf{x} \in \mathfrak{R}^n. \quad (3.1)$$

For this airfoil optimization problem the gradient information is not available and

the gradients are estimated using finite differences. Due to the nature of the problem, a custom finite difference scheme is implemented which records the gradient approximation error. Second order gradient based optimization methods also require the second-derivative (Hessian) of the objective function. The Hessian is also approximated using a finite difference method.

Many gradient methods make use of a line search, which performs a one-dimensional optimization. A custom line search is used for our application. The line search is customized to handle the problem's nature and to allow for task-parallelism. Task-parallelism is where independent/uncoupled tasks are performed by different processors.

The customized finite-difference scheme is discussed first in this chapter followed by the line search section. Take note that the customized methods focus on robustness instead of efficiency, using more evaluations in an attempt to increase the probability of determining a lower function value.

Unconstrained gradient techniques are then discussed, followed by constrained optimization techniques. Among the constrained optimization techniques are off-the-shelf surrogate methods. The chapter concludes with the results from the single objective airfoil optimization.

3.1 Finite differences

Finite difference methods are used to approximate gradients when no direct gradient information is available. Quadratic methods require, in addition to the gradient of the objective function, the constraint gradients and the second derivative (Hessian) of the objective function. For this application all gradients need to be determined numerically.

The finite difference methods implemented are discussed and tested on sample problems. A finite-difference size investigation is then performed on the single objective formulation.

3.1.1 Gradient of the objective function

The finite difference method proposed here determines both the gradient approximation as well as the approximation error. This error is used to select which finite difference size should be used when approximating the airfoil single objective formulation's gradient.

A linear polynomial is fitted to each dimension to determine its gradient component. The polynomial's gradient, b , and its offset, c , are determined by performing a least-square error fit on three sample points. The sample points have function values of y_1 , y_2 and y_3

and offsets of ϵ_1 , ϵ_2 and ϵ_3 , which gives the linear system

$$\begin{bmatrix} \epsilon_1 & 1 \\ \epsilon_2 & 1 \\ \epsilon_3 & 1 \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad (3.2)$$

$$\mathbf{A} \begin{bmatrix} b \\ c \end{bmatrix} = \mathbf{b}. \quad (3.3)$$

The least-square error solution to the over-determined linear system is

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} (\mathbf{A}^T \mathbf{b}), \quad (3.4)$$

which can be expressed as

$$\begin{bmatrix} b \\ c \end{bmatrix} = \frac{1}{3\zeta + \eta^2} \begin{bmatrix} 3 & -\eta \\ -\eta & \zeta \end{bmatrix} \begin{bmatrix} y_1\epsilon_1 + y_2\epsilon_2 + y_3\epsilon_3 \\ y_1 + y_2 + y_3 \end{bmatrix} \quad (3.5)$$

where

$$\zeta = \epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 \quad (3.6)$$

$$\eta = \epsilon_1 + \epsilon_2 + \epsilon_3. \quad (3.7)$$

The gradient approximation presented in (3.5) gives the same gradient values as the well-known central difference approximation when $\epsilon_1 = \epsilon$, $\epsilon_2 = 0$ and $\epsilon_3 = -\epsilon$. The central difference approximation is:

$$b = \frac{(y_1 - y_3)}{2\epsilon}. \quad (3.8)$$

The gradient term of (3.5) is also the same as when a quadratic function is fitted to the three points, with $\epsilon_1 = \epsilon$, $\epsilon_2 = 0$ and $\epsilon_3 = -\epsilon$:

$$\begin{bmatrix} \epsilon^2 & \epsilon & 1 \\ 0 & 0 & 1 \\ \epsilon^2 & -\epsilon & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}, \quad (3.9)$$

giving

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \frac{1}{2\epsilon^3} \begin{bmatrix} \epsilon & -2\epsilon & \epsilon \\ \epsilon^2 & 0 & -\epsilon^2 \\ 0 & 2\epsilon^3 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}. \quad (3.10)$$

As the perturbation size tends to zero the function's behavior becomes linearly dominated, and the true gradient can be obtained (if it exists). This can be illustrated by

examining the Taylor series expansion. The Taylor series expansion of a one-dimensional, infinitely differentiable function f about a point h is

$$f(x) = f(h) + \frac{f'(h)}{1!}(x - h) + \frac{f''(h)}{2!}(x - h)^2 + \frac{f'''(h)}{3!}(x - h)^3 + \dots \quad (3.11)$$

As the spacing for the three sample points decreases, the error for the gradient approximation will decrease. Two forms of linear fit error can be generated, the error-sum \hat{e} and the normalized error \bar{e} , i.e.

$$\hat{e} = \left\| \mathbf{A} \begin{bmatrix} b \\ c \end{bmatrix} - \mathbf{b} \right\| \quad (3.12)$$

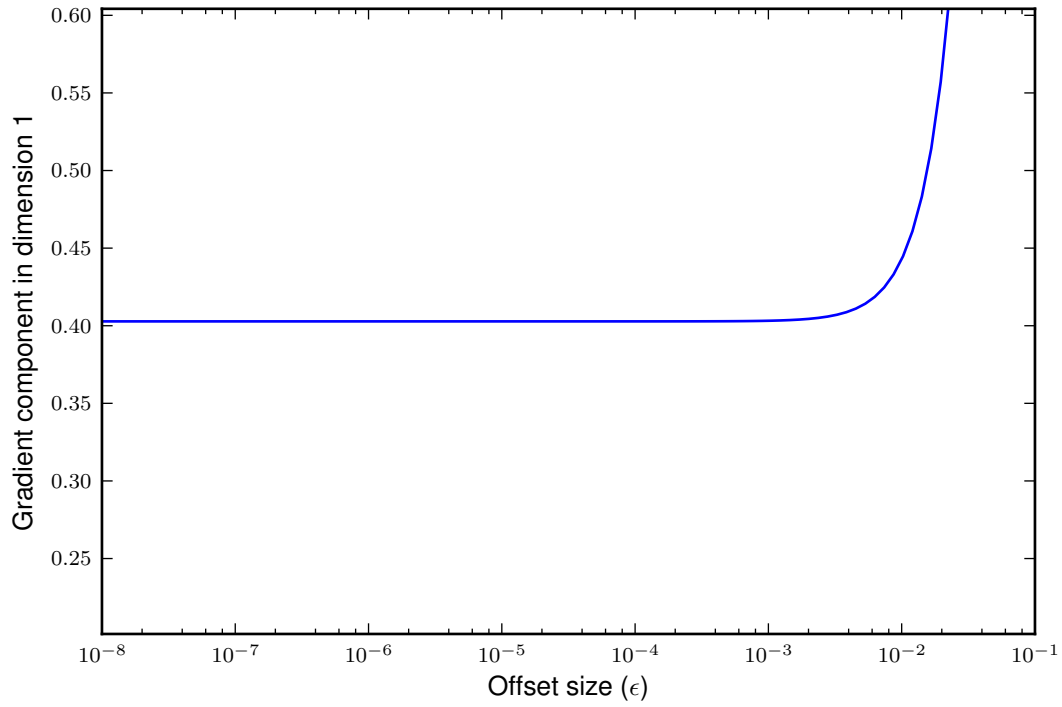
$$\bar{e} = \frac{\hat{e}}{\max(y) - \min(y)}. \quad (3.13)$$

These fit errors are of interest as it gives an indication of how successful the finite difference approximation is. These values are used when deciding which perturbation size to use for the gradient approximations required by the airfoil single objective optimization problem. Figure 3.1 illustrates how the errors decrease for a smooth infinitely-differentiable function.

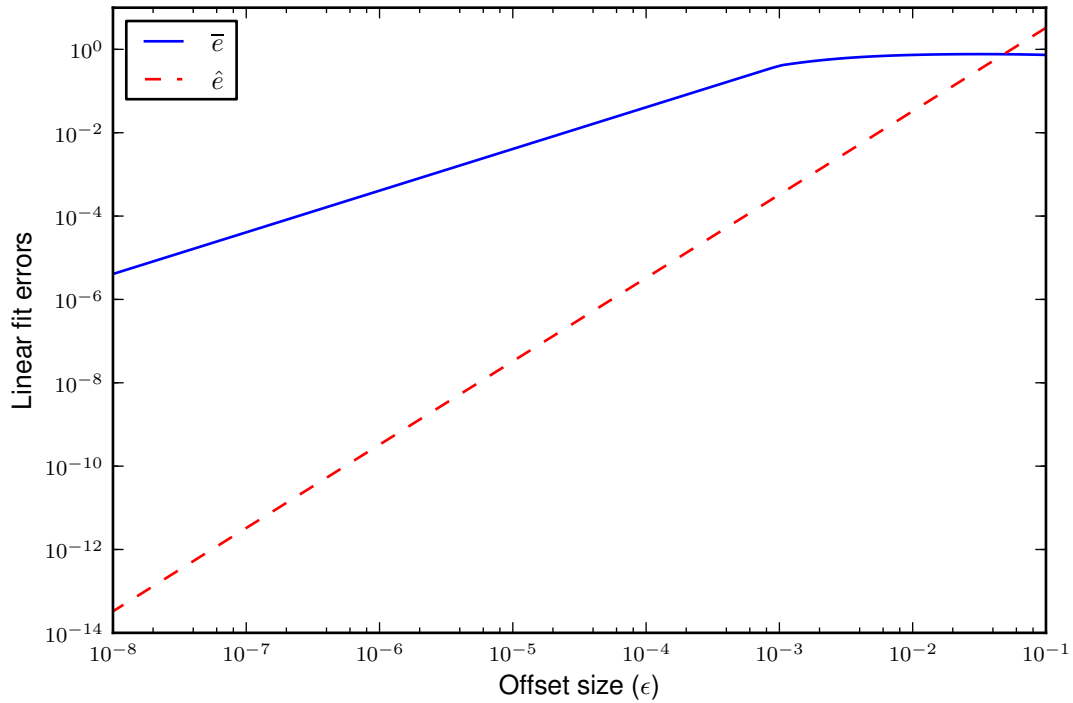
Both error metrics (3.12) and (3.13) should be examined when deciding on the quality of a fit. For example, when a smooth function approaches its minimum and the gradient tends to zero, the function's linear component will approach zero and only its higher order derivatives will affect the sample points leading to a high normalized linear fit error, \bar{e} . But examination of the error-sum \hat{e} will show that the high normalized error is the result of the near-zero linear component.

The least-square error linear fit (3.2) can be used for any combinations of perturbations ϵ , not being limited to evenly spaced sets. This feature is useful if one of the initial sample points has an undefined function value, in which case a new sample point can be generated and used with the old sample points.

$2n + 1$ function evaluations are required to determine f' when a common central point is used. The function evaluations are uncoupled and thus can be computed independently of each other. This allows for Task-parallelism, reducing computing time. The dimension-fit finite difference algorithm is described in Algorithm 1.



(a) Gradient approximation



(b) fit errors

Figure 3.1: (a) Gradient approximation and (b) gradient fit errors of the first dimension of the 2D Rosenbrock Function at $x_1 = 1.001, x_2 = 1.001$ where $f(x_1, x_2) \approx 10^{-4}$.

Algorithm 1 Dimension-fit finite difference pseudo code

```

procedure DIMENSION-FIT FINITE DIFFERENCE( $f, \mathbf{x}, \epsilon$ )
   $y_c = f(\mathbf{x})$  ▷ evaluate common central point
  for  $j \in \{1, 2, \dots, n\}$  do ▷ for each dimension
     $y_1$  ▷ attempt to evaluate forward perturbation for dim.
     $y_3$  ▷ attempt to evaluate backward perturbation for dim.
    if sufficient points generated then
       $b_j$  ▷ fit linear polynomial (3.2)
    else
       $b_j = 0.0$  ▷ ignore dimension
    end if
     $\partial f / \partial x_j = b_j$ 
  end for
end procedure

```

3.1.2 Hessian approximation

The Hessian approximation is excessively expensive, requiring $4n + 2n^2$ function evaluations for the central difference approximations. However a robust Hessian approximation method is desired due to the function's characteristics. Since the custom gradient-based method's focus is on obtaining a better final function value, the large number of function evaluations can be tolerated.

The number of entries (n_h) in the Hessian for n dimensions, is the number of variables plus the number of two-variable combinations that can be made between the variables

$$\begin{aligned}
 n_h &= n + C_2^n \\
 &= n + \frac{n!}{2!(n-2)!} \\
 &= n + \frac{1}{2}n(n-1).
 \end{aligned} \tag{3.14}$$

A central difference fit is used for Hessian approximations. If insufficient points are available for the central differences approximation, forward difference and backward difference are attempted. The following finite differences from [4] are used:

- Central difference approximations for second-order terms,

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{-f(\mathbf{x} + 2\epsilon_i) + 16f(\mathbf{x} + \epsilon_i) - 30f(\mathbf{x}) + 16f(\mathbf{x} - \epsilon_i) - f(\mathbf{x} - 2\epsilon_i)}{12\epsilon^2} \tag{3.15}$$

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{f(\mathbf{x} + \epsilon_i + \epsilon_j) - f(\mathbf{x} + \epsilon_i - \epsilon_j) - f(\mathbf{x} - \epsilon_i + \epsilon_j) + f(\mathbf{x} - \epsilon_i - \epsilon_j)}{4\epsilon^2} \tag{3.16}$$

- forward difference approximations for second-order terms,

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{-f(\mathbf{x} + 2\boldsymbol{\epsilon}_i) - 2f(\mathbf{x} + \boldsymbol{\epsilon}_i) + f(\mathbf{x})}{\epsilon^2} \quad (3.17)$$

$$\frac{\partial^2 f}{\partial x_i x_j} = \frac{f(\mathbf{x} + \boldsymbol{\epsilon}_i + \boldsymbol{\epsilon}_j) - f(\mathbf{x} + \boldsymbol{\epsilon}_i) - f(\mathbf{x} + \boldsymbol{\epsilon}_j) + f(\mathbf{x})}{\epsilon^2} \quad (3.18)$$

- backward difference approximations for second-order terms,

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{-f(\mathbf{x} - 2\boldsymbol{\epsilon}_i) - 2f(\mathbf{x} - \boldsymbol{\epsilon}_i) + f(\mathbf{x})}{\epsilon^2} \quad (3.19)$$

$$\frac{\partial^2 f}{\partial x_i x_j} = \frac{f(\mathbf{x} - \boldsymbol{\epsilon}_i - \boldsymbol{\epsilon}_j) - f(\mathbf{x} - \boldsymbol{\epsilon}_i) - f(\mathbf{x} - \boldsymbol{\epsilon}_j) + f(\mathbf{x})}{\epsilon^2} \quad (3.20)$$

The offset vector $\boldsymbol{\epsilon}_i$, is a zero-vector except for the i 'th dimension where the component is equal to the difference size, ϵ .

3.1.3 Testing

Test functions are used to check that the algorithms are implemented correctly. The finite difference test functions are

$$t_1(x_1, x_2) = (x_1 - 4)^3 + (2 - x_2)^2 \quad (3.21)$$

$$t_2(x_1, x_2) = \begin{cases} t_1(x_1, x_2) & 80\% \text{ of the time} \\ \text{undefined} & \text{otherwise} \end{cases} \quad (3.22)$$

$$t_3(x_1, x_2) = t_1(x_1, x_2) + r(0, 0.001) \quad (3.23)$$

The first test function is uncoupled, with one of the components being cubic and the other quadratic. Since the second component is quadratic the finite-difference should yield the exact solution. The gradient's first component error should decrease as the perturbation size decreases. The second function adds random failure. The third test function adds a noise component to the first test function, with $r(0, 0.001)$ returning a random number between 0 and 0.001 with a uniform probability density.

The gradient approximations of all the test functions are compared to the analytical gradient of t_1 . The analytical gradient of the first test function is

$$t_1'(x_1, x_2) = \begin{bmatrix} 3(x_1 - 4)^2 \\ -2(2 - x_2) \end{bmatrix}. \quad (3.24)$$

Testing functions t_2 and t_3 do not have analytical gradients, but as the test functions

| | $t'_1(3, 3)$ | $t'_2(3, 3)$ | $t'_3(3, 3)$ |
|----------------------|--|--|---|
| $\epsilon = 10^{-1}$ | $\begin{bmatrix} 3.0100 \\ 2.0000 \end{bmatrix}$ | $\begin{bmatrix} 3.0100 \\ 2.0000 \end{bmatrix}$ | $\begin{bmatrix} 3.0101 \\ 2.0015 \end{bmatrix}$ |
| $\epsilon = 10^{-2}$ | $\begin{bmatrix} 3.0001 \\ 2.0000 \end{bmatrix}$ | $\begin{bmatrix} 3.0301 \\ 2.0000 \end{bmatrix}$ | $\begin{bmatrix} 2.9730 \\ 1.9699 \end{bmatrix}$ |
| $\epsilon = 10^{-3}$ | $\begin{bmatrix} 3.0000 \\ 2.0000 \end{bmatrix}$ | $\begin{bmatrix} 3.0000 \\ 2.0000 \end{bmatrix}$ | $\begin{bmatrix} 2.9466 \\ 2.2010 \end{bmatrix}$ |
| $\epsilon = 10^{-6}$ | $\begin{bmatrix} 3.0000 \\ 2.0000 \end{bmatrix}$ | $\begin{bmatrix} 3.0000 \\ 0.0000 \end{bmatrix}$ | $\begin{bmatrix} 24.5554 \\ 109.5661 \end{bmatrix}$ |

Table 3.1: Gradient approximations for various difference sizes

were designed to obscure the first test function’s behavior, they are bench-marked by how close they approximate the gradient of t_1 . Due to the stochastic nature of test functions two and three, their results will change every time the gradient approximations (shown in Table 3.1) are calculated.

When the t_2 function randomly “crashes” it can have two effects on the gradient approximation. If one sample point fails to generate, the accuracy is decreased to that of a forward-difference approximation. If insufficient points are generated, the gradient cannot be estimated, in which case the algorithm returns 0 for that component.

t_3 ’s gradient approximations are of special interest, as the function exhibits noise similar to that of the airfoil single objective cost function. The test demonstrates the negative effect noise has on the gradient approximation as ϵ decreases, when the function changes become dominated by the superimposed noise.

Note that the error in gradient approximation for t_3 forms a probability distribution for a given ϵ . The error’s probability distribution changes for different ϵ , with the mean of the distribution approaching 0 as ϵ decreases, and the probability variance increasing as the ϵ decreases. These effects are illustrated in Figure 3.2 which shows the error distribution generated from 10^5 gradient approximations.

The test functions for the Hessian approximation are:

$$t_4(x_1, x_2) = (x_1 - 4)^3 + (2 - x_2)^2 - x_1x_2^2 \quad (3.25)$$

$$t_5(x_1, x_2) = \begin{cases} t_4(x_1, x_2) & 80\% \text{ of the time} \\ \text{undefined} & \text{otherwise} \end{cases} \quad (3.26)$$

$$t_6(x_1, x_2) = t_4(x_1, x_2) + r(0, 0.001) \quad (3.27)$$

Similarly to the test functions used for the gradient approximation, t_5 and t_6 are designed

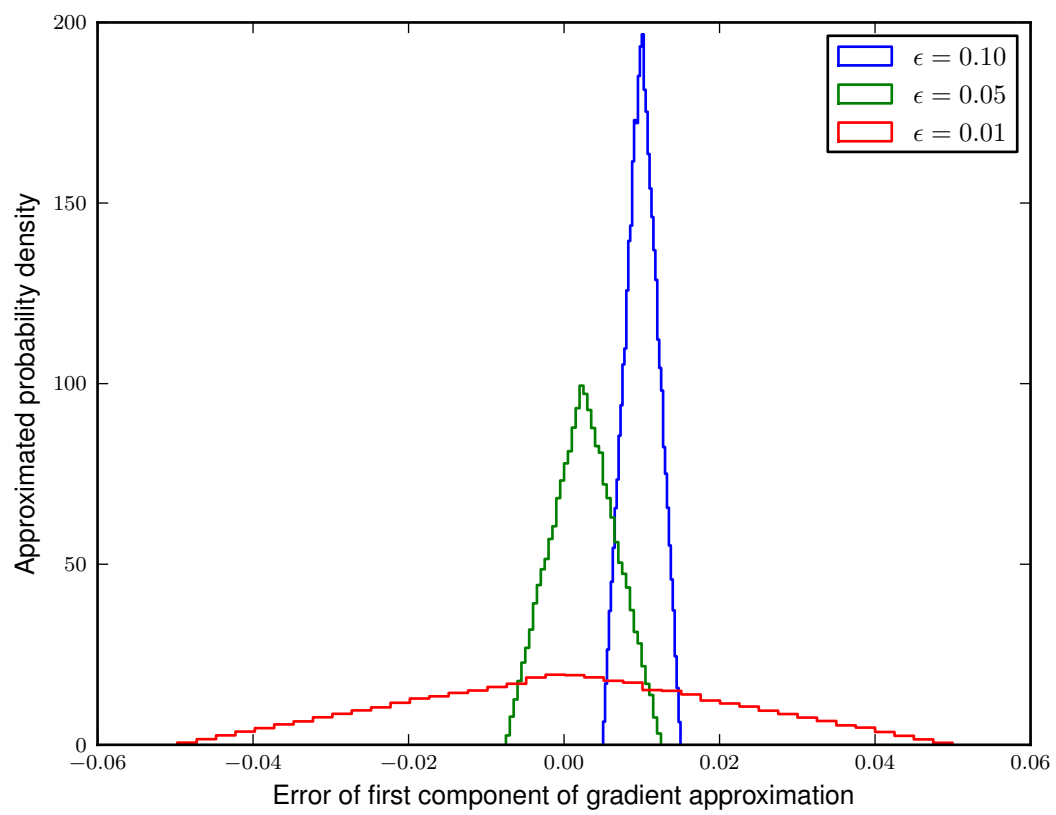


Figure 3.2: The effect of noise on gradient approximations for t_3 .

| | $t'_4(2, -4)$ | $t'_5(2, -4)$ | $t'_6(2, -4)$ |
|----------------------|---|---|---|
| $\epsilon = 10^{-1}$ | $\begin{bmatrix} -12.00 & 8.00 \\ 8.00 & -2.00 \end{bmatrix}$ | $\begin{bmatrix} -12.00 & 8.00 \\ 8.00 & -2.00 \end{bmatrix}$ | $\begin{bmatrix} -12.01 & 8.01 \\ 8.01 & -2.09 \end{bmatrix}$ |
| $\epsilon = 10^{-2}$ | $\begin{bmatrix} -12.00 & 8.00 \\ 8.00 & -2.00 \end{bmatrix}$ | $\begin{bmatrix} -12.00 & 8.00 \\ 8.00 & -2.00 \end{bmatrix}$ | $\begin{bmatrix} -23.36 & 7.68 \\ 7.68 & -6.79 \end{bmatrix}$ |
| $\epsilon = 10^{-3}$ | $\begin{bmatrix} -12.00 & 8.00 \\ 8.00 & -2.00 \end{bmatrix}$ | $\begin{bmatrix} -12.00 & 8.00 \\ 8.00 & -2.00 \end{bmatrix}$ | $\begin{bmatrix} 681.30 & 14.12 \\ 14.12 & -411.98 \end{bmatrix}$ |
| $\epsilon = 10^{-4}$ | $\begin{bmatrix} -12.00 & 8.00 \\ 8.00 & -2.00 \end{bmatrix}$ | $\begin{bmatrix} -12.00 & 8.00 \\ 8.00 & -2.00 \end{bmatrix}$ | $\begin{bmatrix} 139982.32 & 12599.14 \\ 12599.14 & 82128.20 \end{bmatrix}$ |

Table 3.2: The effect of the offset size on Hessian approximations

to obscure t_4 . The Hessian of t_4 is

$$t''_4(x_1, x_2) = \begin{bmatrix} 6(x_1 - 4) & -2x_2 \\ -2x_2 & -2 \end{bmatrix}. \quad (3.28)$$

Despite the central difference oversampling the Hessian approximation is still sensitive to the effect of random noise. Table 3.2 shows the results for the Hessian approximations for a single instance.

3.1.4 Analysis of the single objective cost function's gradient

An investigation is conducted to determine the best choice of finite difference size when optimizing the single objective function. A point in the design space is selected and the gradient value plus the linear dominance errors, for each dimension, are recorded for different values of ϵ .

The results show there is no obvious choice of finite-difference size to use. The linear fit error's minima occur between 10^{-3} and 10^{-1} depending on which dimension is examined. The investigation for dimension 5 is shown in Figure 3.3, and the results for dimension 15 in Figure 3.4.

The investigation further demonstrates that the problem formulations are not well-suited for gradient-based optimization. The error of the linear fit does not decrease with the offset size. The amplitude of the function noise is large and has a significant effect on the gradient approximations.

The avionics box's height constraint behaves well but the maximum lift constraint does not. The numerical noise seems to come from the XFOIL drag analysis, which explains why the height constraint which only depends on the airfoil shape (generated by PROFOIL) does not experience any noise. Figure 3.5 shows the approximation of a component of the height constraint's gradient vector for different ϵ .

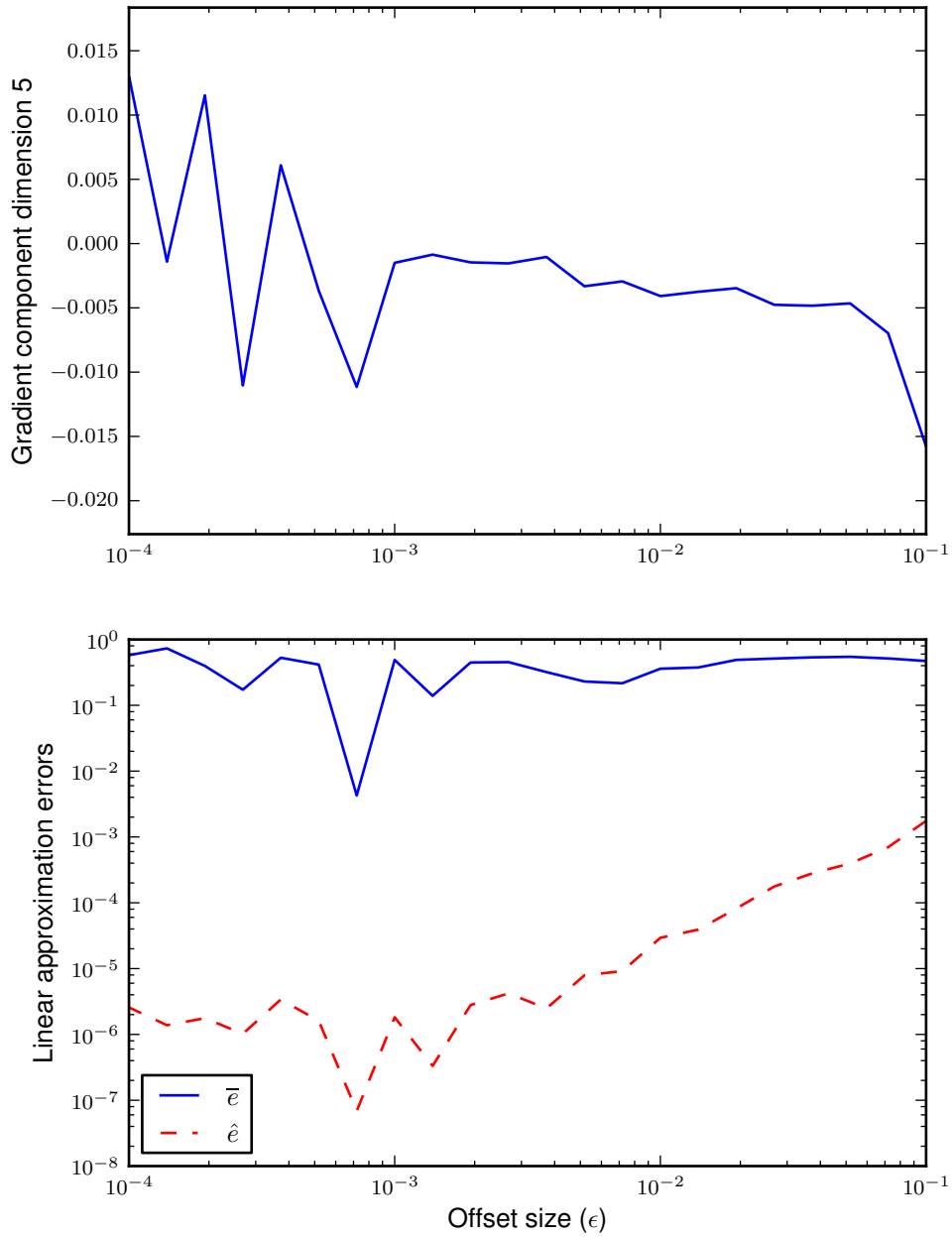


Figure 3.3: Approximation of the 5th component of the gradient of the single objective cost function for different finite difference sizes

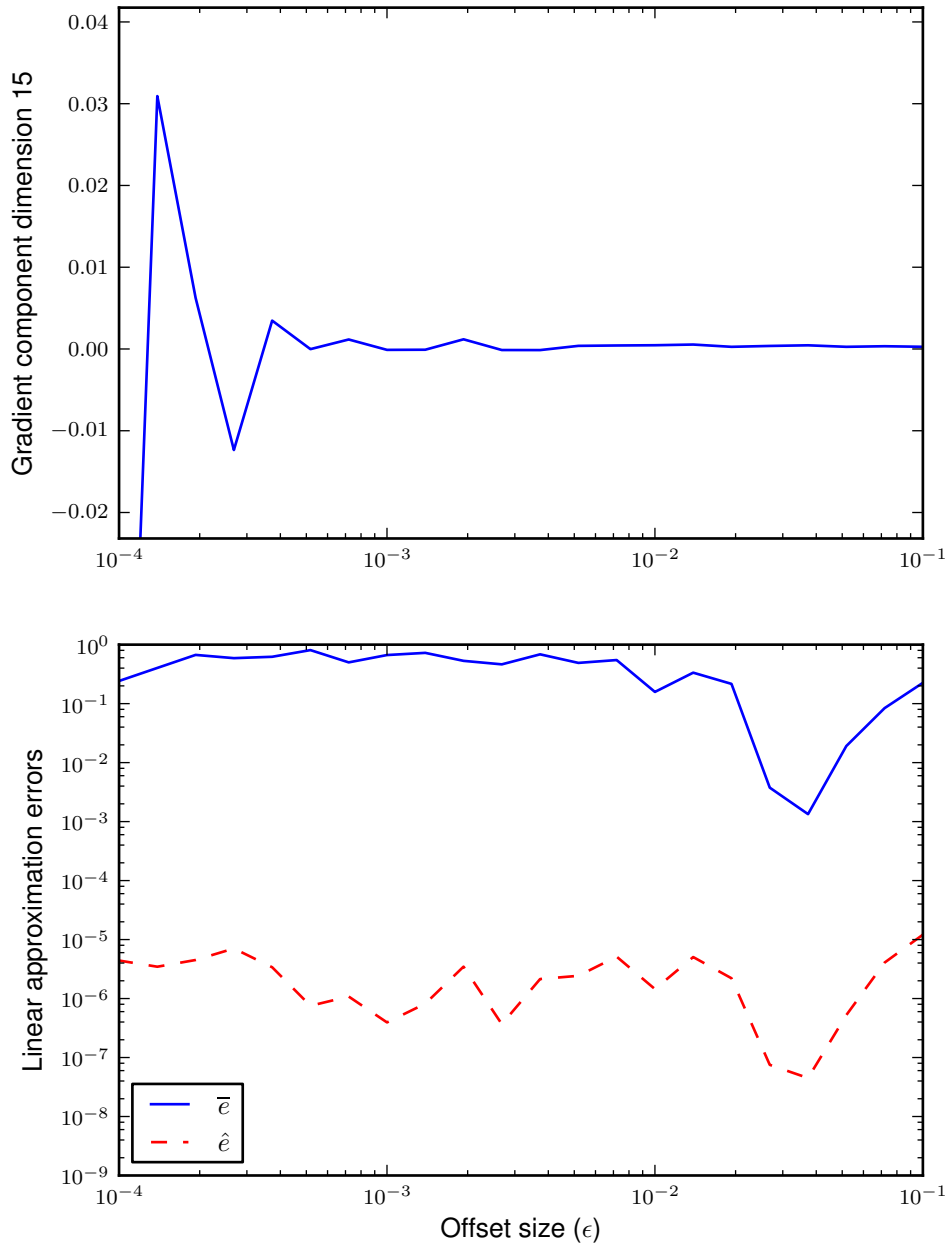


Figure 3.4: Approximation of the 15th component of the gradient of the single objective cost function for different finite difference sizes

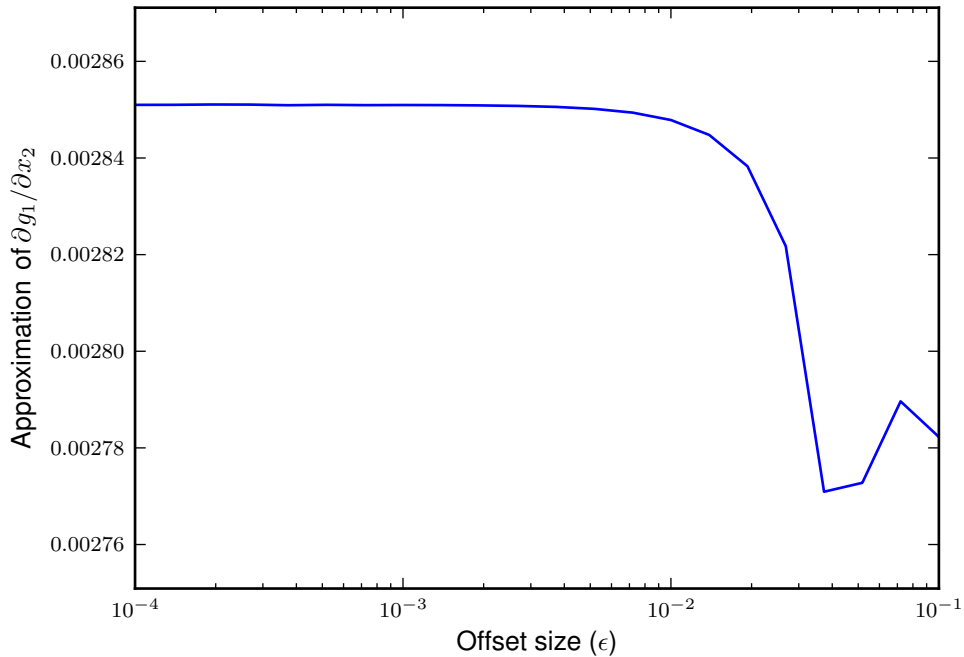


Figure 3.5: Approximation of the second component of the gradient of the avionics box's height for different finite difference sizes

An investigation of the Hessian approximation of the single objective formulation is not done since the Hessian approximation sensitivity to noise is already demonstrated in the testing functions. The noise evident from the gradient finite-difference investigations should compromise the Hessian calculations for any offset size.

Based on the presented results, an ϵ size of 0.05 is chosen for the airfoil optimization gradient approximations. For the Hessian approximation (only used by the SQP method) an ϵ of 0.1 is used. A possibly more accurate approach would be to determine an ϵ for each dimension, but this approach is not investigated here.

3.2 Line Searches

Robust, exact, derivative-free line searches which can handle functions with undefined regions are implemented. Line search methods that require gradients are not considered as function gradient information is unreliable. A line search searches along a direction, α , from a starting point x_i . The line search scalar variable λ maps to the design space as follows:

$$x = x_i + \alpha\lambda. \quad (3.29)$$

Exact and inexact line searches have different termination criteria. Inexact line searches seek only to make an improvement which is deemed acceptable subject to certain

conditions. Exact line searches try to find λ^* as to satisfy:

$$f(\mathbf{x}_i + \boldsymbol{\alpha}\lambda^*) \leq f(\mathbf{x}_i + \boldsymbol{\alpha}\lambda) \text{ for all } \lambda \in \mathfrak{R}. \quad (3.30)$$

Exact methods, for properly conditioned functions, converge to minima but require more function evaluations than inexact methods, which in general decrease the total function evaluations required during the optimization [25]. Inexact methods have different termination criteria, such as the Wolfe conditions or Goldstein conditions [25]. Exact line search methods have been chosen since the problem does not have reliable gradients, which are normally required for the termination criteria of inexact line searches.

The exact line search requirements for the single objective airfoil optimization are:

- Ability to handle non-smooth functions with undefined regions.
- Given an initial search length, the algorithm can extend beyond this area in order to locate a minimum.

A golden section search algorithm and a custom section populating method are investigated. Each method is discussed and then benchmarked on a set of test functions.

3.2.1 Golden section search

The golden section search is a robust method, that refines the line search space until the section size is less than the user specified tolerance. At each line search iteration j , the values at four positions $\lambda_{j,1}$, $\lambda_{j,2}$, $\lambda_{j,3}$ and $\lambda_{j,4}$ are used to approximate the location of the minimum [17]. Given an initial section size Λ_0 , the starting points are

$$\lambda_{0,1} = 0, \quad (3.31)$$

$$\lambda_{0,2} = \phi^2 \Lambda_0, \quad (3.32)$$

$$\lambda_{0,3} = \phi \Lambda_0, \quad (3.33)$$

$$\lambda_{0,4} = \Lambda_0. \quad (3.34)$$

The points are spaced by using the golden ratio, $\phi = -(1 - \sqrt{5})/2$, so that for every search-space refinement only one new function evaluation is required. The section refinement is illustrated in Figure 3.6.

The section size at iteration j , Λ_j reduces at a rate proportional to the golden ratio. The section size when the line search does not extend, as a function of j is

$$\Lambda_j = \phi^j \Lambda_0. \quad (3.35)$$

The extending ability of the line search allows a small Λ_0 to be chosen. This is beneficial in a line search as the majority of the steps are small, with only a few large

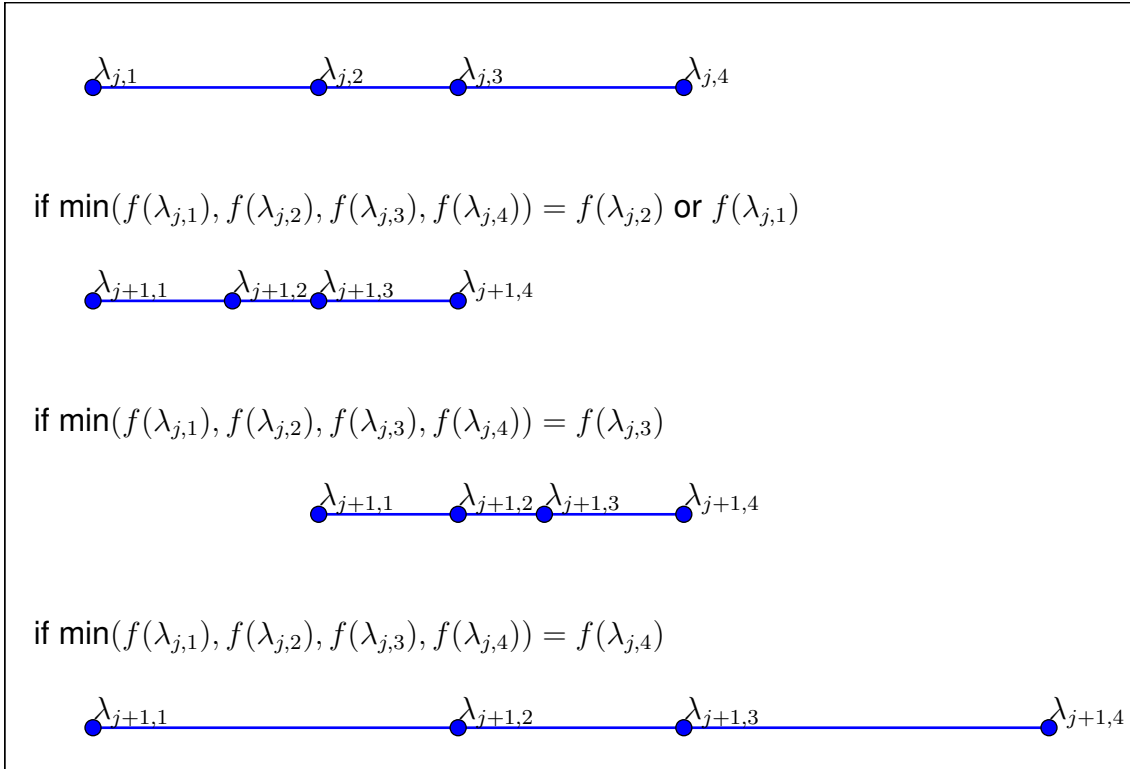


Figure 3.6: Golden Section algorithm progression

steps. If the line search could not extend, two undesirable scenarios can occur. One scenario is when Λ_0 is too small and the minimum cannot be found as it is outside the initial search region. The other scenario is when a large Λ_0 is used, in which case the minimum is more likely to be found but at the cost of extra function evaluations.

When a point in the line search is in undefined space, it is treated as if the function value is larger than a defined point. Since the method only requires function value comparisons between different points, this is sufficient to handle the function failure scenario.

The termination criteria is the section size limit δ_x . The number of iterations j_{\max} is determined by using δ_x and the initial section size Λ_0 as follows:

$$j_{\max} = \left\lceil \frac{\log(\delta_x/\Lambda_0)}{\log \phi} \right\rceil. \quad (3.36)$$

3.2.2 Section populating line search

The section populating line search is a custom method, developed here by the author for multi-modal line functions with undefined regions. The method focuses on robustness over efficiency, making use of over sampling in order to escape local minima. The over sampling and intelligent point selection increases the probability that the line function's minimum will be determined. Its advantages over the golden section method is that it is

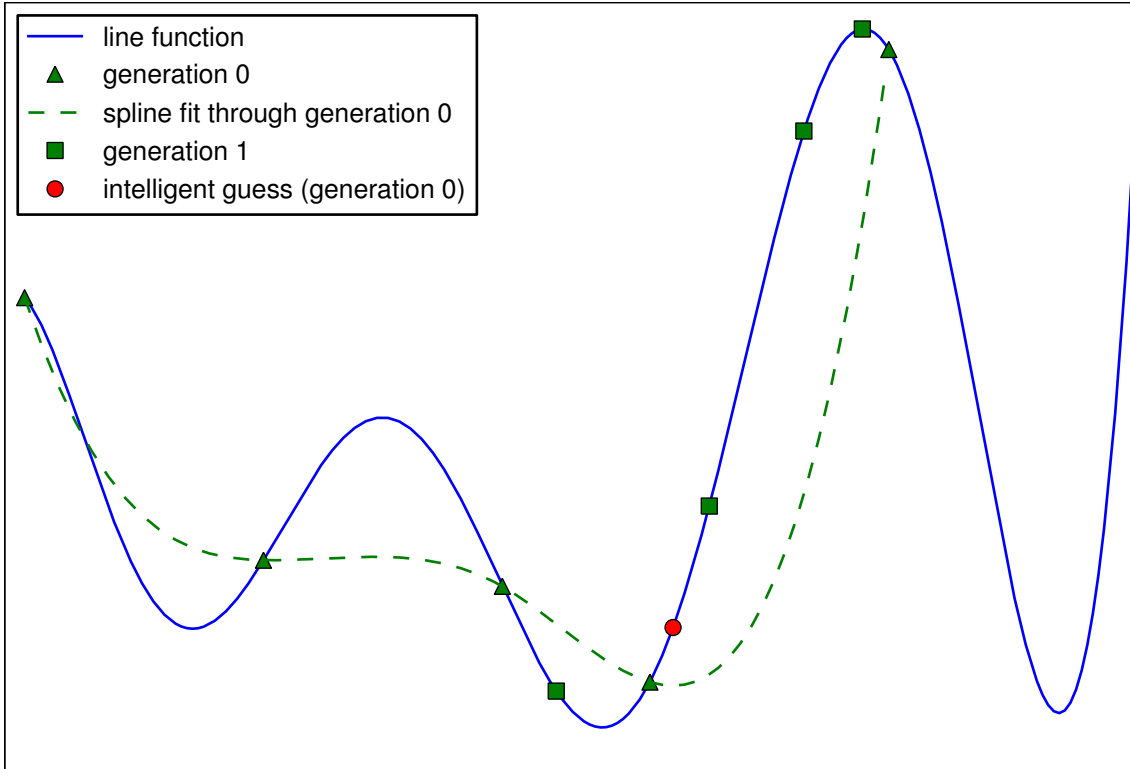


Figure 3.7: Process of point selection used by section populating line search

more robust and it allows for task parallelism.

The first iteration involves populating the initial section with N points. The algorithm then uses a spline to approximate where the minimum is. The suspected minimum, together with further populating points (in a refined search area) are then evaluated. This process is repeated until either the maximum number of iterations is exceeded or the solution appears to be within the required tolerance. The process is illustrated in Figure 3.7.

After each iteration, j , the search section is refined. The search bounds for the current iteration are the lower bound, $\lambda_{j,l}$, and the upper bound $\lambda_{j,u}$. The minimum evaluation location so far λ_k , is used to determine the refined section. The index k is the index of minimum function value in a list sorted by λ values. The new bounds are then chosen to be the evaluations adjacent to λ_k . Should the minimum point be at the largest λ so far, the search area is expanded by a factor β . Summarizing the above,

$$\lambda_{j+1,l} = \begin{cases} \lambda_{k-1} & \text{if } k > 0 \\ \lambda_0 & \text{otherwise,} \end{cases} \quad (3.37)$$

$$\lambda_{j+1,u} = \begin{cases} \lambda_{k+1} & \text{if } \lambda_k < \max(\boldsymbol{\lambda}) \\ \lambda_k + \beta(\lambda_{j,u} - \lambda_{j,l}) & \text{otherwise.} \end{cases} \quad (3.38)$$

The populating points are generated using the golden ratio as a basis. An irrational basis is chosen, as this will ensure that no two points will have the same λ value. The modulus of the golden ratio multiplied by points generated count q , is used as the seed generating new sampling points,

$$\lambda = \lambda_{j,l} + \text{mod}(q\phi, 1)(\lambda_{j,u} - \lambda_{j,l}) \text{ where } q \in 0, 1, 2, \dots \quad (3.39)$$

After the initial generation, intelligent point selection is implemented. This involves taking the points from the previous generation and fitting a response surface to them. The minimum of the response surface is then assumed to be an improved approximation of the actual minimum location. In this case, a 3rd-degree piece-wise polynomial is used as the response surface. The m 'th section of the spline p^m , defined between λ_σ^m and λ_τ^m is expressed as

$$p^m(z) = a_0^m + a_1^m z^1 + a_2^m z^2 + a_3^m z^3, \quad (3.40)$$

where

$a_0^m, a_1^m, a_2^m, a_3^m$ – spline coefficients

m – the piece-wise polynomial index

z – co-ordinate from origin of piece-wise polynomial, $\lambda - \lambda_\sigma^m$

Enforcing that the piecewise polynomial value is equal to the function value at its origin y_m , and continuity between the gradient and second derivative, gives:

$$p^m(0) = y_m, \quad (3.41)$$

$$p^m(\lambda_\tau^m - \lambda_\sigma^m) = p^{m+1}(0), \quad (3.42)$$

$$\frac{\partial p^m}{\partial \lambda}(\lambda_\tau^m - \lambda_\sigma^m) = \frac{\partial p^{m+1}}{\partial \lambda}(0), \quad (3.43)$$

$$\frac{\partial^2 p^m}{\partial \lambda^2}(\lambda_\tau^m - \lambda_\sigma^m) = \frac{\partial^2 p^{m+1}}{\partial \lambda^2}(0). \quad (3.44)$$

The continuity conditions leave two unknowns, as the continuity of the spline's gradient and second derivative cannot be applied at the ends. If the natural spline formulation [8] is applied, then the second derivative at both ends are set to zero. If the clamped spline [8] boundary conditions are applied, then the gradient at both ends are specified.

For this implementation, the four data points closest together, are merged into one part of the piecewise polynomial. This merged part consists of four points, and hence all the coefficients for this piece can be solved directly using conventional curve fitting routines. The rest of the pieces are then determined using (3.41) - (3.44).

The maximum number of iterations, j_{\max} , is determined by approximating when the section size will be below a specified tolerance δ_x . The section size Λ_j is the difference

between the current boundaries:

$$\Lambda_j = \lambda_{j,u} - \lambda_{j,l}. \quad (3.45)$$

We define the reduction ratio r_j as the proportion between two consecutive refinements:

$$r_j = \frac{\Lambda_j}{\Lambda_{j-1}}. \quad (3.46)$$

If the intelligent point selection process completely fails, then the section size after j iterations is:

$$\Lambda^j = \Lambda_0 \prod_{i=1}^j r_i, \quad (3.47)$$

$$\Lambda^j \approx \Lambda_0 (2/(N+1))^j. \quad (3.48)$$

Thus the maximum number of iterations is:

$$j_{max} = \text{ceil} \left(\frac{\log(\delta_x/\Lambda_0)}{\log(2/(N+1))} \right). \quad (3.49)$$

After each iteration the minimum point evaluated so far, is recorded. If the change in λ between two consecutive points is less than δ_x the search is terminated. The only exception to this is if the minimum point is at the starting point, in which case the search will not be terminated. This is important as it eliminates premature terminations.

At generation 0, N populating points are evaluated. For each generation after that $N - 1$ populating points and one intelligent point are evaluated. For parallel computing, task parallelism allows each iteration to be split into N tasks, each which can be handled by worker processes.

3.2.3 Testing

The two line search techniques are tested on a set of test functions. This is done to check that the algorithms are coded correctly and to confirm that the algorithms meet the design criteria mentioned earlier in this section.

A quadratic one-dimensional function is used as the first test function t_1 , with later test functions having additional complexity added-on. The second test function t_2 has discontinuities in the gradient information and is multi-modal. The third test function t_3 contains undefined regions. The fourth test function t_4 is a combination of t_2 and t_3 .

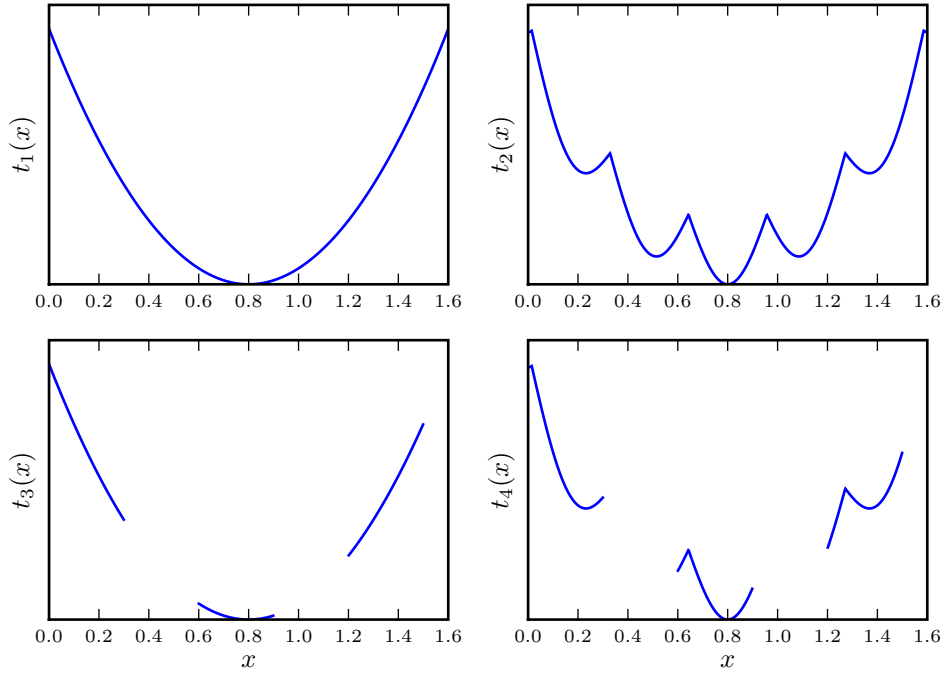


Figure 3.8: Line search test functions

Figure 3.8 depicts the functions used. The test functions are:

$$t_1(x) = (x - 0.8)^2 \quad (3.50)$$

$$t_2(x) = t_1(x) - 0.2 \sin(\text{mod}(10(x - 0.8) + \pi/2), \pi) \quad (3.51)$$

$$t_3(x) = \begin{cases} t_1(x) & \text{if } \text{mod}(x, 0.6) > 0.3 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (3.52)$$

$$t_4(x) = \begin{cases} t_2(x) & \text{if } \text{mod}(x, 0.6) > 0.3 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (3.53)$$

The starting point for each of the test functions is 0, and the number of populating points for the section populating search is 10. The section size limit is 10^{-6} , and section initial size Λ_0 is 1.0. The minimum returned from the algorithms and the number of evaluations made for all of the test functions is shown in Table 3.3.

Both line searches are implemented correctly, with the minimum being found in every case except one. The golden section search failed to determine the minimum of t_2 as the method is developed under the assumption that an uni-modal function is minimized. Since t_2 is multi-modal, it is possible to generate sequences that reduce the section incorrectly. Further detail on the line search performances, including the golden section failure, can be found in Appendix A.

| | Golden Section | | Section Populating | |
|-------|----------------|-------------|--------------------|-------------|
| | evals. | λ^* | evals. | λ^* |
| t_1 | 36 | 0.800293 | 30 | 0.799844 |
| t_2 | 33 | 0.854102 | 50 | 0.800000 |
| t_3 | 36 | 0.800293 | 30 | 0.800000 |
| t_4 | 36 | 0.800293 | 50 | 0.800000 |

Table 3.3: The number of function evaluations used (evals.) and the approximate minimum location found (λ^*) by the implemented line searches on the set of test functions

The section populating line search is used for the airfoil single objective optimization. The line search algorithm is more robust compared to the golden section search. It also allows for task parallelism since the populating points evaluated every generation of the line search are independent of each other. More than one processor can therefore be used in the line search of the sequential airfoil single objective optimization problem.

3.3 Unconstrained optimization methods

Unconstrained optimization methods, used in conjunction with the Augmented Lagrangian, are implemented in the airfoil optimization. The primary text from which these algorithms were implement is *Practical Mathematical Optimization* [32]. The unconstrained optimizers applied are

- BFGS - a quasi-newton method,
- CGPR - a conjugate gradient method with Polak-Ribiere search directions and
- LFOP - a dynamic particle trajectory method.

The unconstrained optimization problem is to determine the location of the function minimum \mathbf{x}^* ,

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \text{ for all } \mathbf{x} \in \mathfrak{R}^n. \quad (3.54)$$

Provided that the line searches and gradient functions can handle the undefined regions in the design space, so should the BFGS and CGPR methods. “Handle” in this context does not imply “perform as if there where no undefined regions” but rather, run without crashing. Section 3.3.3 discusses the LFOP method and the work-around implemented to allow the method to handle undefined regions.

3.3.1 BFGS

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) method is a quasi-newton method which makes use of an approximated Hessian to minimize a function. After each iteration the inverse of the Hessian is updated using a rank-2 update. The inverse Hessian is used to determine the line search directions.

At a local minimum $\hat{\mathbf{x}}$, the gradients are zero and the Hessian f'' , is positive-definite.

$$f'(\hat{\mathbf{x}}) = \mathbf{0}, \quad (3.55)$$

$$\mathbf{y}^T f''(\hat{\mathbf{x}}) \mathbf{y} > 0 \text{ for any } \mathbf{y} \in \mathbf{R}^n, f''(\hat{\mathbf{x}}) \in \mathbf{R}^{n \times n}. \quad (3.56)$$

Newton methods use the Hessian and the gradient information to determine a corrective step \mathbf{x}_c to approach the minimum. At each iteration i , the design \mathbf{x}_i is therefore updated by

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{x}_c, \quad (3.57)$$

with \mathbf{x}_c determined to solve the subproblem,

$$f''(\mathbf{x}_i) \mathbf{x}_c + f'(\mathbf{x}_i) = \mathbf{0}. \quad (3.58)$$

The corrective step, \mathbf{x}_c , is used as a line search direction as this increases stability. The BFGS method does not determine the Hessian directly (which is very expensive if not directly available), but uses a second-order update method to approximate the Hessian inverse \mathbf{G}_i . The update equation is given by

$$\mathbf{G}_i = \mathbf{G}_{i-1} + \left[1 + \frac{\mathbf{y}_i^T \mathbf{G}_{i-1} \mathbf{y}_i}{\mathbf{v}_i^T \mathbf{y}_i} \right] \left[\frac{\mathbf{v}_i \mathbf{v}_i^T}{\mathbf{v}_i^T \mathbf{y}_i} \right] - \left[\frac{\mathbf{v}_i \mathbf{y}_i^T \mathbf{G}_{i-1} + \mathbf{G}_{i-1} \mathbf{y}_i \mathbf{v}_i^T}{\mathbf{v}_i^T \mathbf{y}_i} \right] \quad (3.59)$$

where

$$\mathbf{v}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$$

$$\mathbf{y}_i = f'(\mathbf{x}_i) - f'(\mathbf{x}_{i-1}).$$

The termination criteria implemented are a step limit, δ_x , and maximum allowable number of iterations i_{\max} . The method is outlined in Algorithm 2.

3.3.2 CGPR

The conjugate gradient methods use mutually conjugate search directions during optimization. The method that is implemented makes use of Polak-Ribiere search directions. Consecutive exact line searches using mutually conjugate search directions, will converge to the exact minimum of a quadratic function, in a number of steps less than or equal to

Algorithm 2 BFGS pseudo code

```

procedure BFGS( $\mathbf{x}_0, i_{max}, \delta_x$ )
   $\mathbf{G} = \mathbf{I}$  ▷ use identity matrix for first  $G$ 
  for  $i \in \{1, 2, \dots, i_{max}\}$  do
    if  $i > 1$  then
       $\mathbf{G}_i$  ▷ (3.59)
    end if
     $\mathbf{u}_i = -\mathbf{G}f'(\mathbf{x}_i)$  ▷ new search direction
     $\mathbf{x}_i = \mathbf{x}_{i-1} + \lambda^*\mathbf{u}_i$  ▷ where  $\lambda^*$  minimizes  $f(\mathbf{x}_{i-1} + \lambda\mathbf{u}_i)$ 
    if termination criteria met then exit
  end for
end procedure

```

the number of dimensions of the function.

The Polak-Ribiere search directions \mathbf{u}_i at each iteration i after the first, is calculated using the following update rule

$$\mathbf{u}_i = -f'(\mathbf{x}_{i-1}) + \mathbf{u}_{i-1} \left(\frac{(f'(\mathbf{x}_{i-1}) - f'(\mathbf{x}_{i-2}))^T f'(\mathbf{x}_{i-1})}{\|f'(\mathbf{x}_{i-1})\|^2} \right). \quad (3.60)$$

The method has the same termination criteria as the BFGS method, namely a step-size limit and a maximum number of iterations. The CGPR method is described in Algorithm 3.

Algorithm 3 CGPR pseudo code

```

procedure CGPR( $\mathbf{x}_0, i_{max}, \delta_x$ )
  for  $i \in \{1, 2, \dots, i_{max}\}$  do
    if  $i > 1$  then
       $\mathbf{u}_i$  ▷ next search direction (3.60)
    else
       $\mathbf{u}_i = -f'(\mathbf{x}_{i-1})$ 
    end if
     $\mathbf{x}_i = \mathbf{x}_{i-1} + \lambda^*\mathbf{u}_i$  ▷ where  $\lambda^*$  minimizes  $f(\mathbf{x}_{i-1} + \lambda\mathbf{u}_i)$ 
    if termination criteria met then exit
  end for
end procedure

```

When testing the conjugate gradient methods on quadratic test functions, it should be noted that exact gradients and line searches are required to converge to a minimum in a number of steps less than or equal to the dimension of the test problem.

3.3.3 LFOP

The leap-frog method for unconstrained optimization (LFOP) is a dynamic trajectory method. The method is robust for applications where the objective functions contain

noise [31]. A particle's dynamics is simulated, with the negative gradient of the objective function used to accelerate the particle towards the function minimum.

LFOP requires only function gradient evaluations, and does not make use of line searches. The particle's acceleration $\ddot{\mathbf{x}}_i$ at iteration i , is the negative gradient of the objective function at its current position \mathbf{x}_i ,

$$\ddot{\mathbf{x}}_i = -f'(\mathbf{x}_i). \quad (3.61)$$

The particle's acceleration and velocity ($\dot{\mathbf{x}}_i$) are integrated numerically over a time step Δ_t . The particle's velocity and position for next iteration $i + 1$ is therefore,

$$\dot{\mathbf{x}}_{i+1} = \dot{\mathbf{x}}_i + \ddot{\mathbf{x}}_i \Delta_t \quad (3.62)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \dot{\mathbf{x}}_i \Delta_t. \quad (3.63)$$

When the euclidean norm of the particle's velocity increases, an interference strategy is implemented to remove energy from the system. The velocity and position are then altered as follows

$$\dot{\mathbf{x}}_{i+1} = \frac{1}{4} (\dot{\mathbf{x}}_{i+1} + \dot{\mathbf{x}}_i) \quad (3.64)$$

$$\mathbf{x}_{i+1} = \frac{1}{2} (\mathbf{x}_{i+1} + \mathbf{x}_i). \quad (3.65)$$

The method contains other heuristics to govern parameters such as the time step, and the termination criteria. Further information on the heuristics can be found in *Figure 4.1* of [32].

The airfoil optimization formulations contain large undefined regions which the LFOP algorithm cannot handle without modification. Two approaches to handle the undefined-regions are

1. continue to travel through the undefined region, hoping to come out in defined space
or
2. steer the particle back towards defined space.

Which approach will work best depends on the problem. In this work, if the particle ends up in undefined space, a distance based function is used to generate an artificial gradient to steer the particle towards its last defined point. The artificial function P , depends on the last defined design found \mathbf{x}_j and is stated as

$$P = \zeta \sum_{k=1}^n (x_{i,k} - x_{j,k})^2. \quad (3.66)$$

The gradient P' is then given by

$$P' = \frac{\partial P}{\partial \mathbf{x}} = 2\zeta(\mathbf{x}_i - \mathbf{x}_j). \quad (3.67)$$

ζ is chosen to normalize the gradient of P such that

$$\|P'(\mathbf{x}_i)\| = \rho_u \|f'(\mathbf{x}_j)\| \quad (3.68)$$

$$\zeta = \frac{\rho_u \|f'(\mathbf{x}_j)\|}{2\|\mathbf{x}_i - \mathbf{x}_j\|}. \quad (3.69)$$

The undefined penalty factor ρ_u is problem specific and determines how aggressive the particle is steered away from the undefined space. If ρ_u is chosen less than zero the particle will be attracted to the undefined region, and hence will try to push through. ρ_u should not be chosen close to zero, as this activates the gradient-norm termination criteria. The modified acceleration rule used in place of (3.61) is

$$\ddot{\mathbf{x}}_i = \begin{cases} -f'(\mathbf{x}_i) & \text{if } \mathbf{x}_i \text{ is defined} \\ -P'(\mathbf{x}_i) & \text{otherwise} \end{cases} \quad (3.70)$$

The leap-frog algorithm for constrained optimization (LFOPC) is discussed in Section 3.4.1. The testing of the unconstrained gradient-based algorithms follows in the next subsection.

3.3.4 Testing

Five standard test functions are used to check that the methods are implemented correctly. The test functions together with their starting points and optima are

- Rosenbrock's parabolic valley:

$$t_1(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \quad (3.71)$$

Starting point $[-1.2, 1.0]$, minimum at $[1, 1]$.

- Quadratic function:

$$t_2(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2. \quad (3.72)$$

Starting point $[0, 0]$, minimum at $[1, 3]$.

- Powell's quartic function:

$$t_3(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4. \quad (3.73)$$

Starting point $[3, -1, 0, 1]$, minimum at $[0, 0, 0, 0]$.

- Fletcher and Powell's helical valley:

$$f(x_1, x_2, x_3) = 100(x_3 - 10\theta(x_1, x_2))^2 + 100 \left(\sqrt{x_1^2 + x_2^2} - 1 \right)^2 + x_3^2, \quad (3.74)$$

$$\text{where } 2\pi\theta(x_1, x_2) = \begin{cases} \arctan \frac{x_2}{x_1} & \text{if } x_1 > 0 \\ \pi + \arctan \frac{x_2}{x_1} & \text{if } x_1 < 0 \end{cases}. \quad (3.75)$$

Starting point $[-1, 0, 0]$, minimum at $[1, 0, 0]$.

- Freudenstein and Roth function:

$$f(x_1, x_2) = (-13 + x_1 + ((5 - x_2)x_2 - 2)x_2)^2 \quad (3.76)$$

$$+ (-29 + x_1 + ((x_2 + x_1)x_2 - 14)x_2)^2. \quad (3.77)$$

Starting point $[-0.5, -2]$, local minimum at $[11.414141, -0.8968]$, minimum at $[5, 4]$.

The optimization methods are tested with the step tolerance and finite difference size set to 10^{-6} . The section populating search was used for the line searches with a populating size of 10, and the dimension-fit finite difference method was used to approximate the gradients. The number of function evaluations and minima returned by the unconstrained gradient-based algorithms are shown in Table 3.4.

The BFGS and CGPR methods should find the minimum of the quadratic test function in two or less iterations. They each take three iterations (50-60 evaluations per iteration) but actually locate the minima after the second iteration, with the final iteration occurring due the termination criteria used.

All the methods converge to the minimum of the convex test functions. The performance of the CGPR function on t_3 , Powell's quartic function, is poor but the method finds the minimum given enough iterations.

For the Freudenstein and Roth function t_5 , all the methods converge to the function's local minimum, not the global minimum. Although the methods are shown to converge to the minimum of a convex function, they are unlikely to converge to the global minimum of a multi-modal function such as t_5 .

Table 3.4: The number of function evaluations and the minima returned, for the implemented unconstrained gradient-based optimization techniques on the test functions.

| | BFGS | | CGPR | | LFOP | |
|-------|-------|---|-------|---|-------|---|
| | evals | xMin | evals | xMin | evals | xMin |
| t_1 | 1065 | $\begin{bmatrix} 1.0000 \\ 1.0000 \end{bmatrix}$ | 1095 | $\begin{bmatrix} 1.0000 \\ 1.0000 \end{bmatrix}$ | 1040 | $\begin{bmatrix} 1.0001 \\ 1.0002 \end{bmatrix}$ |
| t_2 | 155 | $\begin{bmatrix} 1.0000 \\ 3.0000 \end{bmatrix}$ | 155 | $\begin{bmatrix} 1.0000 \\ 3.0000 \end{bmatrix}$ | 600 | $\begin{bmatrix} 1.0000 \\ 3.0000 \end{bmatrix}$ |
| t_3 | 1574 | $\begin{bmatrix} -0.0000 \\ 0.0000 \\ -0.0000 \\ -0.0000 \end{bmatrix}$ | 4314 | $\begin{bmatrix} 0.0001 \\ -0.0000 \\ -0.0002 \\ -0.0002 \end{bmatrix}$ | 2403 | $\begin{bmatrix} -0.0068 \\ 0.0007 \\ -0.0000 \\ -0.0000 \end{bmatrix}$ |
| t_4 | 1174 | $\begin{bmatrix} 1.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}$ | 2147 | $\begin{bmatrix} 1.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}$ | 1701 | $\begin{bmatrix} 1.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}$ |
| t_5 | 355 | $\begin{bmatrix} 11.4128 \\ -0.8968 \end{bmatrix}$ | 365 | $\begin{bmatrix} 11.4128 \\ -0.8968 \end{bmatrix}$ | 1295 | $\begin{bmatrix} 11.4116 \\ -0.8969 \end{bmatrix}$ |

It is rare that optimization problem have no constraints. The next section deals with the optimization of constrained functions using gradient-based techniques.

3.4 Constrained gradient-based methods

The constrained optimization problem is expressed as follows: Minimize an objective function f , such that all inequality constraints \mathbf{g} and equality constraints \mathbf{h} are satisfied:

$$\begin{aligned}
 &\text{minimize} && f(\mathbf{x}) \\
 &\text{such that} && \mathbf{h}(\mathbf{x}) = \mathbf{0} \\
 &&& \text{and} && \mathbf{g}(\mathbf{x}) \leq \mathbf{0}.
 \end{aligned} \tag{3.78}$$

The constrained gradient-based methods used on the single objective airfoil optimization problem are

- LFOPC - Leap-Frog Algorithm for Constrained Optimization.
- LMM - Lagrangian Multiplier Methods, in conjunction with the unconstrained gradient-based techniques discussed in the previous section.
- SQP - Sequential Quadratic Programming method.

Surrogate methods [5] were also used on the *a priori* single objective airfoil optimization formulations. The methods considered here construct non-local approximations to

the objective and constraints and sequentially solve the approximate problems. Since the approximations are non-local and do not require gradient evaluations they are less affected by the presence of noise. The surrogate based methods used on the single objective airfoil optimization problem are

- COBYLA - Constrained Optimization BY Linear Approximation, applied without modification directly from the Scipy Python package [2].
- SLSQP - Sequential Least Squares Programming optimization algorithm applied without modification directly from the Scipy Python package[2].

3.4.1 LFOPC

The constrained version for the leap-frog optimizer (LFOPC) is implemented in a similar manner as it was presented by Snyman [31]. The method consists of three phases, in which three separate unconstrained optimization problems are formulated and optimized using the LFOP algorithm.

The first unconstrained formulation F_0 , used for phase 0, makes use of the weak penalty-factor ρ_0 giving

$$F_0(\mathbf{x}) = f(\mathbf{x}) + \rho_0 \left(\mathbf{h}(\mathbf{x})^T \mathbf{h}(\mathbf{x}) + \langle \mathbf{g}(\mathbf{x}) \rangle^T \langle \mathbf{g}(\mathbf{x}) \rangle \right), \quad (3.79)$$

with the “ $\langle \cdot \rangle$ ” operator defined as

$$\langle \mathbf{a} \rangle = \max(a_i, 0) \text{ for } i \in \{1, 2, \dots, n\}. \quad (3.80)$$

The second LFOP optimization continues from the optimum found in phase 0 ($\hat{\mathbf{x}}_0$) but on a stricter penalty formulation $\rho_1 \gg \rho_0$. The second sub-problem used in phase 1, F_1 , is

$$F_1(\mathbf{x}) = f(\mathbf{x}) + \rho_1 \left(\mathbf{h}(\mathbf{x})^T \mathbf{h}(\mathbf{x}) + \langle \mathbf{g}(\mathbf{x}) \rangle^T \langle \mathbf{g}(\mathbf{x}) \rangle \right). \quad (3.81)$$

The final unconstrained problem F_2 is used to approximate the function optimum \mathbf{x}^* . The last phase involves satisfying the active constraints \mathbf{g}_a . The active constraints are determined by examining the constraint values at the solution to F_1 , $\hat{\mathbf{x}}_1$, which should be close to the true solution. All the inequality constraints violated at $\hat{\mathbf{x}}_1$ form \mathbf{g}_a .

$$g_a \in g \text{ where } \mathbf{g}(\hat{\mathbf{x}}_1) > 0. \quad (3.82)$$

F_2 uses only constraint information and no objective function information,

$$F_2(\mathbf{x}) = \rho_1 \left(\mathbf{h}(\mathbf{x})^T \mathbf{h}(\mathbf{x}) + \mathbf{g}_a(\mathbf{x})^T \mathbf{g}_a(\mathbf{x}) \right). \quad (3.83)$$

In this implementation the step size parameter passed to the LFOP algorithm δ , is different for each phase. This is as the average solution update required by the optimizer in each iteration decreases for each phase. Given a starting point \mathbf{x}_0 the following relation normally holds

$$\|\mathbf{x}_0 - \hat{\mathbf{x}}_0\| \gg \|\hat{\mathbf{x}}_0 - \hat{\mathbf{x}}_1\| \geq \|\hat{\mathbf{x}}_1 - \mathbf{x}^*\|. \quad (3.84)$$

A heuristic is used to determine the step size parameter to pass to the LFOP optimizer for each phase. It depends on the initial step size parameter δ_0 and the step tolerance ϵ_x .

$$\delta_0 \text{ user specified} \quad (3.85)$$

$$\delta_1 = \min(\delta_0, 50\epsilon_x) \quad (3.86)$$

$$\delta_2 = \delta_1 \quad (3.87)$$

The LFOPC pseudo code is shown in Algorithm 4.

Algorithm 4 LFOPC pseudo code

```

procedure LFOPC( $\mathbf{x}_0, \rho_0, \rho_1, \delta_0, \delta_1$ )
   $\hat{\mathbf{x}}_0 = \text{LFOP}(F_0, \mathbf{x}_0, \delta_0)$  ▷ minimize  $F_0$  using LFOP, starting at  $\mathbf{x}_0$ 
   $\hat{\mathbf{x}}_1 = \text{LFOP}(F_1, \hat{\mathbf{x}}_0, \delta_1)$ 
   $\hat{\mathbf{x}}^* = \text{LFOP}(F_2, \hat{\mathbf{x}}_1, \delta_1)$ 
end procedure

```

3.4.2 Lagrange multiplier method

The classical Lagrange multiplier method [32] solves the constrained optimization problem by creating a Lagrangian function L such that the minimum of this unconstrained function occurs at the minimum of the original problem. A Lagrangian has equality constraint multipliers $\boldsymbol{\mu}$, and inequality constraint multipliers $\boldsymbol{\lambda}$. The Lagrangian is

$$L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{g}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{h}(\mathbf{x}). \quad (3.88)$$

The Lagrangian multipliers $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ need to be chosen as to satisfy the definition of the constrained optimization problem (3.78). The inequality multipliers are zero for the inactive inequality constraints at the solution. The challenge when using the classical method is to determine what Lagrangian weights are required. The Lagrangian problem

is

$$\text{minimize } L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) \quad (3.89)$$

$$\text{such that } \frac{\partial L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})}{\partial \boldsymbol{\mu}} = \mathbf{0} \quad (3.90)$$

$$\text{and } \frac{\partial L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}} = \mathbf{0}. \quad (3.91)$$

The penalty formulation of the constrained optimization problem adds a penalty to the function value related to the amount by which the constraints are violated. A typical penalty formulation is given below,

$$P(\mathbf{x}) = f(\mathbf{x}) + \rho \left(\mathbf{h}(\mathbf{x})^T \mathbf{h}(\mathbf{x}) + \langle \mathbf{g}(\mathbf{x}) \rangle^T \langle \mathbf{g}(\mathbf{x}) \rangle \right). \quad (3.92)$$

The amount by which the constraints are violated at the solution to $P(\mathbf{x})$ depends on the magnitude of the penalty factor ρ , if the constraints are active at the minimum. As the value of ρ increase the closer the minimum of the penalty formulation comes to the original problem. The gradients also become ill-conditioned as the penalty factor increases.

The penalty formulation is easy to implement, but struggles to obtain a solution with high accuracy on the constraints without creating an ill-conditioned function. The Lagrangian function's minimum is the same as the constrained problem minimum, given that multiplier weights can be accurately determined.

The augmented Lagrangian techniques [32], or Lagrange multiplier methods (LMMs), make use a combination of a penalty and a Lagrangian formulation. The method makes use of successive approximations of the Lagrangian multipliers $\hat{\boldsymbol{\mu}}_j$ and $\hat{\boldsymbol{\lambda}}_j$ to estimate the true Lagrangian multipliers. The augmented Lagrangian is as follows

$$\hat{L}_j(\mathbf{x}) = f(\mathbf{x}) + \hat{\boldsymbol{\mu}}_j^T \mathbf{h}(\mathbf{x}) + \rho \mathbf{h}(\mathbf{x})^T \mathbf{h}(\mathbf{x}) + \rho \left\langle \frac{\hat{\boldsymbol{\lambda}}_j}{2\rho} + \mathbf{g}(\mathbf{x}) \right\rangle^T \langle \mathbf{g}(\mathbf{x}) \rangle. \quad (3.93)$$

When the augmented Lagrangian multipliers have the same values as the true Lagrange multipliers, the minimum of the function will be at the solution to the constrained optimization problem \mathbf{x}^* , i.e.

$$\min \hat{L}_j(\mathbf{x}) \approx \min L(\mathbf{x}). \quad (3.94)$$

The augmented Lagrangian multipliers are updated by examining the stationary point

of the function. The gradient at the minimum of the augmented Lagrangian $\hat{\mathbf{x}}_j^*$ is

$$\frac{\partial \hat{L}_j}{\partial \hat{\mathbf{x}}_j^*} = \frac{\partial f}{\partial \hat{\mathbf{x}}_j^*} + \left\langle \hat{\boldsymbol{\lambda}}_j^T + 2\rho \right\rangle \mathbf{g}'(\hat{\mathbf{x}}_j^*) + (\hat{\boldsymbol{\mu}}_j^T + 2\rho) \mathbf{h}'(\hat{\mathbf{x}}_j^*) = 0 \quad (3.95)$$

and the gradient for the Lagrangian is

$$\frac{\partial L}{\partial \hat{\mathbf{x}}_j^*} = \frac{\partial f}{\partial \hat{\mathbf{x}}_j^*} + \boldsymbol{\lambda}^T \mathbf{g}'(\hat{\mathbf{x}}_j^*) + \boldsymbol{\mu}^T \mathbf{h}'(\hat{\mathbf{x}}_j^*) \approx 0. \quad (3.96)$$

The multipliers are updated so that the penalty function will not have an effect on the gradient. At the solution to the constrained optimization problem, no constraints are violated and hence there will be no penalty. Hence the method for updating the multipliers of the augmented Lagrangian is

$$\hat{\boldsymbol{\mu}}_{j+1} = \hat{\boldsymbol{\mu}}_j + 2\rho \mathbf{h}(\hat{\mathbf{x}}_j^*) \quad (3.97)$$

$$\hat{\boldsymbol{\lambda}}_{j+1} = \left\langle \hat{\boldsymbol{\lambda}}_j + 2\rho \mathbf{g}(\hat{\mathbf{x}}_j^*) \right\rangle. \quad (3.98)$$

The multiplier method hence consists of consecutive unconstrained optimizations, where any unconstrained optimizer can be used. The method continues to update the augmented Lagrangian until the solution of the constrained optimization problem is obtained. The method terminates when either the maximum iteration count is exceeded, or when the solution from the augmented Lagrangian returns a valid solution. The pseudo code for the Lagrange multiplier method (LMM) is shown in Algorithm 5.

Algorithm 5 LMM pseudo code

procedure LMM(\mathbf{x}_0)

$$\hat{\boldsymbol{\mu}}_1 = 0$$

▷ first optimize pure penalty function

$$\hat{\boldsymbol{\lambda}}_1 = 0$$

for $j = 1, 2, \dots, j_{max}$ **do**

determine $\hat{\mathbf{x}}_j^*$, by minimizing \hat{L}_j

$$\hat{\boldsymbol{\mu}}_{j+1} = \hat{\boldsymbol{\mu}}_j + 2\rho \mathbf{h}(\hat{\mathbf{x}}_j^*)$$

▷ update multipliers

$$\hat{\boldsymbol{\lambda}}_{j+1} = \left\langle \hat{\boldsymbol{\lambda}}_j + 2\rho \mathbf{g}(\hat{\mathbf{x}}_j^*) \right\rangle$$

Exit if all constraints are satisfied

end for

end procedure

3.4.3 SQP

The Sequential Quadratic Programming (SQP) method solves successive quadratic programming problems in order to solve the constrained optimization problem. The quadratic programming (QP) problem determines the minimum of a quadratic function consisting

of a Hessian \mathbf{A} , and plane component \mathbf{b} , such that the linear equality and linear inequality constraints are satisfied i.e.

$$\begin{aligned} & \text{minimize} && \frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} \\ & \text{such than} && \mathbf{C} \mathbf{x} + \mathbf{d} = \mathbf{0} \\ & \text{and} && \mathbf{E} \mathbf{x} + \mathbf{f} \leq \mathbf{0} \end{aligned} \tag{3.99}$$

where

$$\begin{aligned} \mathbf{C} \in \mathfrak{R}^{n \times k} & \quad \text{– the equality constraint gradient} \\ \mathbf{E} \in \mathfrak{R}^{n \times j} & \quad \text{– the inequality constraint gradient} \\ n & \quad \text{– number of design variables} \\ k & \quad \text{– number of equality constraints} \\ j & \quad \text{– number of inequality constraints.} \end{aligned}$$

In this implementation an “off-the-shelf” QP solver from the Python package, Cvxopt [13] is used. For each quadratic programming problem, a quadratic representation of the function and linear representation of the constraints are generated.

The SQP method moves from its starting point, \mathbf{x}_0 , by solving successive QP problems. The optimizer’s current location \mathbf{x}_i and the solution to the QP problem \mathbf{x}_q are used to determine the line search direction at each iteration. The search direction at the i ’th iteration, \mathbf{u}_i is

$$\mathbf{u}_i = \mathbf{x}_q - \mathbf{x}_i. \tag{3.100}$$

The line searches are done on an augmented Lagrangian (3.93), with multipliers taken from the QP problem’s solution. A line search is used instead of moving directly to the solution to the QP problem to increase stability.

In the event that the QP programming fails, the line search is performed on the penalty formulation. The penalty function is:

$$P(\mathbf{x}) = f(\mathbf{x}) + \rho \left(\mathbf{h}(\mathbf{x})^T \mathbf{h}(\mathbf{x}) + \langle \mathbf{g}(\mathbf{x}) \rangle^T \langle \mathbf{g}(\mathbf{x}) \rangle \right). \tag{3.101}$$

The line search direction \mathbf{u}_i is chosen in the steepest descent direction:

$$\mathbf{u}_i = -P' \tag{3.102}$$

$$P' = \frac{\partial P}{\partial \mathbf{x}} = f'(\mathbf{x}) + 2\rho \mathbf{h}^T(\mathbf{x}) \mathbf{h}(\mathbf{x}) + 2\rho \frac{\partial}{\partial \mathbf{x}} \langle \mathbf{g}(\mathbf{x}) \rangle^T \langle \mathbf{g}(\mathbf{x}) \rangle. \tag{3.103}$$

Substituting the QP problem approximations of the objective function and constraints

functions gives the steepest descent search direction as

$$P' = \mathbf{b} + 2\rho(\mathbf{C}_a^T \mathbf{d}_a + \mathbf{E}^T \mathbf{f}), \quad (3.104)$$

with \mathbf{C}_a and \mathbf{d}_a consisting of the active inequality constraints only.

The termination criteria for this SQP method are a minimum step-size and maximum number of iterations. The pseudo code for the SQP method is given in Algorithm 6.

Algorithm 6 SQP pseudo code

```

procedure SQP( $\mathbf{x}_0$ )
  for  $i = 1, 2, \dots, i_{max}$  do
    if QP problem has a solution then
       $\mathbf{u}_i$  ▷ (3.100)
       $\mathbf{x}_i = \mathbf{x}_{i-1} + \lambda^* \mathbf{u}_i$  ▷ where  $\lambda^*$  minimizes  $\hat{L}(\mathbf{x}_i + \lambda \mathbf{u}_i)$ 
    else
       $\mathbf{u}_i$  ▷ use steepest descent (3.104)
       $\mathbf{x}_i = \mathbf{x}_{i-1} + \lambda^* \mathbf{u}_i$  ▷ where  $\lambda^*$  minimizes  $P(\mathbf{x}_i + \lambda \mathbf{u}_i)$ 
    end if
    if termination criteria met then exit
  end for
end procedure

```

3.4.4 COBYLA

Constrained Optimization BY Linear Approximation (COBYLA) is a successive linear programming algorithm. It is a derivative free method developed for nonlinearly constrained functions by M.J.D. Powell [27]. During each iteration linear approximations of the objective function and constraints are constructed. The linear approximations are used to determine an update step through linear programming.

A simplex of $n + 1$ vertexes is used for the linear approximations. The change in the simplex is governed by the solution of a linear programming problem, and the quality of its shape. A trust-region is used to control the changes of variables recommend from the solution to the linear programming problem. The vertexes are also adjusted in order to improve the simplex shape.

The starting value of the trust region size τ , is decreased during the optimization until it is smaller than a desired trust region size, at which stage the optimization terminates. The user specifies the initial trust region size τ_i and desired final trust region size τ_f .

The method has the desirable property that it does not require smoothness, but it does require that the entire design space be defined. The work around implemented is that if the objective function or constraints are undefined, then they would be assigned

a value larger than any in the design space. The modified single objective function f_c is

$$f_c = \begin{cases} f(\mathbf{x}) & \text{if } f(\mathbf{x}) \text{ defined} \\ 0.06 & \text{otherwise,} \end{cases} \quad (3.105)$$

and the modified constraint violation function g_c , is

$$g_c = \begin{cases} -g(\mathbf{x}) & \text{if } g(\mathbf{x}) \text{ defined} \\ -0.1 & \text{otherwise} \end{cases} \quad (3.106)$$

The COBYLA algorithm is used “off-the-shelf” from the Scientific Tools for Python package[2]. g_c is the negative of g due the constraint sign convention in Scipy.

3.4.5 SLSQP

The sequential least square quadratic programming (SLSQP) algorithm [21] uses a modified version of the non negative least squares (NNLS) algorithm from [22] as its core routine. The NNLS routine is used to solve the generalized nonlinear optimization problem as described in [22].

Successive quadratic problems are solved using non localized approximations of the optimization problem. Second-order approximations of the objective function are made using BFGS second order updates and first-order approximations are made of the constraint functions.

The undefined design work around and modified constraint functions are applied as described in the COBYLA subsection.

3.4.6 Testing

Problems from [19] were used to test the implementations of the constrained-optimization algorithms. The summary of results from eight test problems are shown in Table 3.5. The complete set of test functions used can be found in Appendix B.

The first test problem, t_1 , is the Rosenbrock function with only one constraint that is not active at the minimum. Every optimization routine managed to solved this effectively unconstrained problem except for the COBYLA method. This method struggles to travel through the valley, terminating after 17000 evaluations with a distance of 0.0016 from the optimum.

The fourth test problem, t_4 , is an uncoupled cubic function of two variables, with two inequality constraints both of which are active at the optimum. All the optimization methods successfully determine the minimum.

| | COBYLA | LFOPC | LMM-BFGS | LMM-CGPR | LMM-LFOP | SQP | SLSQP |
|-----------|--------------------|-------------------|------------------|-------------------|----------|-------------------|------------------|
| t_1 | 17391 _x | 1281 | 1296 | 1196 | 1856 | 885 | 78 _x |
| t_4 | 17 ^v | 2313 | 552 ^v | 562 ^v | 6552 | 239 | 8 |
| t_{10} | 84 ^v | 2338 _x | 3552 | 2442 ^v | - | - | 45 ^v |
| t_{25} | - | - | - | - | - | - | - |
| t_{100} | 355 ^v | - | - | - | - | 6843 ^v | 124 ^v |
| t_{113} | 337 ^v | - | - | - | - | 3312 | 147 ^v |

The number of function evaluations required to obtain the solution \mathbf{x} , with $\|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-4}$, $\max(g(\mathbf{x})) \leq 0$ and $\max(|h(\mathbf{x})|) < 10^{-4}$.

^v the constraint criteria is loosely satisfied: $0 < \max(g(\mathbf{x})) \leq 10^{-6}$ or $10^{-4} < \max|h(\mathbf{x})| < 10^{-2}$

^x the distance criteria is loosely satisfied : $10^{-4} < \|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-2}$

- the returned solution is outside the required tolerances.

Table 3.5: Testing results of the constrained optimization algorithms.

The tenth test function, t_{10} , consists of a linear objective function with a quadratic inequality constraint. Note that the LMM-LFOP method returns a solution with a distance 0.007 from the optimum after about 1900 iterations.

All the algorithms were run on the test batch with the same settings. t_{25} is an example where the penalty factor, which worked well on the majority of functions, traps the optimizer at its starting point. If the penalty factor is reduced, all the optimizers can find the solution. Penalty factor sensitivity is also present in the single objective airfoil formulations.

The t_{100} test problem consists of seven variables and four inequality constraints. The LMM methods terminate falsely due to the step-limit termination criteria. This has a detrimental effect as it leads to incorrect Lagrange multipliers being selected. This happens as the LMM method assumes that the unconstrained optimizer terminates at the augmented Lagrangian, and updates the Lagrangian multipliers accordingly. The LFOPC method gets close to the solution but not within the required tolerances.

The t_{113} test problem has ten variables and eight constraints. Similar to t_{100} , only the SQP method, SLSQP method and the COBYLA methods are able to determine the solution. The LMM method's sub-problems are not solved correctly, and the LFOPC method returns a solution that is outside the required tolerances.

All the methods can, given the correct settings, determine the test problem's optima. The testing results for the entire test-set are located in Appendix C. The implemented methods are able to handle the test functions. In the next section their performance on the single objective airfoil optimization formulations are presented.

3.5 Airfoil optimization

The gradient-based methods implemented and the surrogate methods are applied to the two different single objective formulations from multiple starting points. The earlier investigation into the model's characteristics indicate that the prerequisites for gradient-based optimization are not met. But all the methods are still expected to improve the function value and the customized methods have been modified in order to enhance their performance on noisy partially defined functions. The results indicate the success of these modifications, and give insight into the applicability of gradient-based method for these types of optimization formulations.

The two *a priori* airfoil formulations are:

1. The avionics box height, or fixed volume variant,

$$f(\mathbf{x}) = 3C_{D_1}(\mathbf{x}) + C_{D_2}(\mathbf{x}) + C_{D_3}(\mathbf{x}), \quad (3.107)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \text{maxLift}(\mathbf{x}) \\ 73\text{mm} - B_h(\mathbf{x}) \end{bmatrix}. \quad (3.108)$$

where maxLift is the maximum lift coefficient of the airfoil and B_h is the height available for an unrotated avionics box.

2. The maximum lift variant,

$$f(\mathbf{x}) = 3C_{D_1}(\mathbf{x}) + C_{D_2}(\mathbf{x}) + C_{D_3}(\mathbf{x}), \quad (3.109)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \text{maxLift}(\mathbf{x}) \\ 10\text{mm} - B_h(\mathbf{x}) \end{bmatrix}. \quad (3.110)$$

Each optimization algorithm is benchmarked from the same ten starting points which are distributed around the design space. The starting points \mathbf{x}_c were generated by first attempting to generate 150 candidate designs. Each design's dimension values are generated using random numbers uniformly distributed between 0 and 1.

From those initial points only 20 points were defined and valid. The final points were then selected to generate the most distributed set. This was achieved by rejecting the candidate points with the highest crowding value. The crowding value for each point c_i is determined using,

$$c_i = \sum_{j=1, j \neq i}^{20} \frac{1}{(\mathbf{x}_c^j - \mathbf{x}_c^i)^T (\mathbf{x}_c^j - \mathbf{x}_c^i)}. \quad (3.111)$$

The elimination process involves discarding the most crowded point until the desired set size is achieved. The crowding values are recalculated after each point is discarded.

The majority of the settings used are the same as those in the constrained optimization testing section. The most significant is the increased finite-difference size as to reduce

| sp | f_0 | evals. < 500 | | evals. < 2000 | | 2000 < evals. | |
|----|---------------|--------------|---------------|---------------|---------------|---------------|---------------|
| | | method | f_{min} | method | f_{min} | method | f_{min} |
| 1 | 0.0571 | SLSQP | 0.0563 | LMM-BFGS | 0.0396 | LMM-LFOP | 0.0367 |
| 2 | 0.0552 | CObyLA | 0.0382 | CObyLA | 0.0382 | CObyLA | 0.0382 |
| 3 | 0.0548 | SLSQP | 0.0493 | LMM-BFGS | 0.0391 | LMM-LFOP | 0.0369 |
| 4 | 0.0532 | SLSQP | 0.0532 | LMM-CGPR | 0.0482 | LMM-LFOP | 0.0466 |
| 5 | 0.0561 | SLSQP | 0.0561 | CObyLA | 0.0385 | LMM-LFOP | 0.0370 |
| 6 | 0.0589 | CObyLA | 0.0416 | CObyLA | 0.0416 | LMM-LFOP | 0.0366 |
| 7 | 0.0611 | CObyLA | 0.0541 | LMM-CGPR | 0.0491 | LMM-LFOP | 0.0452 |
| 8 | 0.0600 | CObyLA | 0.0545 | CObyLA | 0.0545 | CObyLA | 0.0545 |
| 9 | 0.0572 | CObyLA | 0.0437 | CObyLA | 0.0437 | LMM-LFOP | 0.0365 |
| 10 | 0.0539 | CObyLA | 0.0394 | LMM-CGPR | 0.0388 | LMM-BFGS | 0.0374 |

Table 3.6: Best Optimization Results from various starting points (sp) for the avionics box height formulation.

the effect of the noise. The gradient finite difference size is increased to $\epsilon = 0.05$, and the step-size termination criteria of $\delta_x = 0.001$ is used. The algorithm specific settings are

- COBYLA: initial trust region $\tau_i = 0.1$, maximum function evaluations 10 000.
- LFOPC: undefined space avoidance factor $\rho_u = 1.0$. 500 iterations per phase.
- LMM-BFGS: BFGS $I_{max} = 150$, LMM It = 6, $\rho = 15.0$
- LMM-CGPR: same as *LMM-BFGS*
- LMM-LFOP: LFOP $I_{max} = 500$, LMM It = 6, $\rho = 50.0$, initial step 0.05, undefined space avoidance factor $\rho_u = 1.0$
- SQP: Hessian finite difference perturbation size $\epsilon = 0.1$, line search penalty function parameter $\rho = 100.0$

The customized methods, in general achieved lower function values than the “off-the-shelf” surrogate methods but also required more function evaluations. The results from the avionics box height formulation are summarized in Table 3.6 and the results from the maximum lift formulation are summarized in Table 3.7.

The COBYLA and the SLSQP off-the-shelf surrogate methods which performed well in testing failed to significantly decrease the function value for the avionics box height formulation. The off-the-shelf methods terminated after few function evaluations at a cost function value higher than those achieved by the customized methods. For the COBYLA method in the maximum lift formulation it is the opposite, with the method in general being more expensive but returning lower function values.

| sp | f_0 | evals. < 500 | | evals. < 2000 | | 2000 < evals. | |
|----|---------------|--------------|---------------|---------------|---------------|---------------|---------------|
| | | method | f_{min} | method | f_{min} | method | f_{min} |
| 1 | 0.0571 | LMM-CGPR | 0.0300 | LMM-CGPR | 0.0300 | SQP | 0.0296 |
| 2 | 0.0552 | CObyLA | 0.0299 | LMM-CGPR | 0.0266 | LMM-CGPR | 0.0266 |
| 3 | 0.0548 | LMM-CGPR | 0.0356 | LMM-BFGS | 0.0299 | SQP | 0.0260 |
| 4 | 0.0532 | LMM-CGPR | 0.0302 | LMM-CGPR | 0.0302 | LMM-CGPR | 0.0302 |
| 5 | 0.0561 | LMM-BFGS | 0.0392 | CObyLA | 0.0258 | CObyLA | 0.0258 |
| 6 | 0.0589 | LMM-BFGS | 0.0281 | LMM-CGPR | 0.0258 | SQP | 0.0247 |
| 7 | 0.0611 | CObyLA | 0.0278 | CObyLA | 0.0278 | CObyLA | 0.0278 |
| 8 | 0.0600 | CObyLA | 0.0545 | CObyLA | 0.0545 | CObyLA | 0.0545 |
| 9 | 0.0572 | LMM-BFGS | 0.0312 | LMM-BFGS | 0.0312 | LMM-BFGS | 0.0312 |
| 10 | 0.0539 | SLSQP | 0.0391 | LMM-CGPR | 0.0249 | LMM-CGPR | 0.0249 |

Table 3.7: Best Optimization Results from various starting points (sp) for the maximum lift formulation.

The results suggest that certain methods are well-suited for rough estimates due to their economy. These rough estimations should not however be expected to return a solution near to the minimum of the airfoil formulations.

The COBYLA method results are presented in Table 3.8. The best run (lowest cost function value with no constraint violation) and the most expensive run for the avionics box height formulation began at the first starting point. The best run for the maximum lift formulation begins from the tenth point. Besides the first run for the avionics box height formulation, the number of function evaluations are low. The method does not always return valid designs for the avionics box height formulation, with half of the designs returning minor constraint violations.

The SLSQP method is the most economical of the methods, each time terminating in under 100 function evaluations. The method does not achieve a large function value decrease though. Table 3.9 shows the SLSQP results.

The LMM-BFGS and the LMM-CGPR optimization results for avionics box height formulation are comparable to each other. The number of function evaluation ranges from over one-hundred to almost 7000. Both methods performed well from the 5th starting point, obtaining a function value of 0.0373 in about 4000 function evaluations.

For the maximum lift formulation the LMM-CGPR method performed better than the LMM-BFGS method. The LMM-BFGS method returned 0.0249 from the tenth starting point which is the second lowest values obtained by the gradient-based methods. The results from LMM-BFGS methods are shown in Table 3.10 and the results from the LMM-CGPR in Table 3.11.

The SQP method could not solve any QP problems for the avionics box height formulation and hence performed poorly. This is because the determined Hessian matrices

| Avionics box height formulation | | | | Maximum lift formulation | | | |
|---------------------------------|---------------|----------------|------------|--------------------------|---------------|----------------|------------|
| sp [‡] | f | V [†] | evals. | sp [‡] | f | V [†] | evals. |
| 1 | 0.0391 | 0 | 2025 | 1 | 0.0346 | 0 | 272 |
| 2 | 0.0382 | $< 10^{-4}$ | 315 | 2 | 0.0299 | 0 | 295 |
| 3 | 0.0402 | 0 | 514 | 3 | 0.0304 | 0 | 960 |
| 4 | 0.0449 | 0.0002 | 220 | 4 | | | 132 |
| 5 | 0.0385 | $< 10^{-4}$ | 545 | 5 | 0.0258 | 0 | 784 |
| 6 | 0.0416 | 0 | 359 | 6 | 0.0284 | 0 | 254 |
| 7 | 0.0541 | $< 10^{-4}$ | 208 | 7 | 0.0278 | 0 | 362 |
| 8 | 0.0545 | 0 | 156 | 8 | 0.0545 | 0 | 156 |
| 9 | 0.0437 | 0 | 281 | 9 | | | 245 |
| 10 | 0.0394 | $< 10^{-4}$ | 390 | 10 | 0.0258 | 0 | 553 |

When the optimiser returns a design which is undefined, the row is left blank.

[‡] starting point

[†] maximum constraint violation

Table 3.8: COBYLA Optimization Results

| Avionics box height formulation | | | | Maximum lift formulation | | | |
|---------------------------------|---------------|----------------|-----------|--------------------------|---------------|----------------|-----------|
| sp [‡] | f | V [†] | evals. | sp [‡] | f | V [†] | evals. |
| 1 | 0.0563 | 0 | 19 | 1 | 0.0563 | 0 | 19 |
| 2 | 0.0552 | 0 | 28 | 2 | 0.0552 | 0 | 28 |
| 3 | 0.0493 | 0 | 76 | 3 | 0.0493 | 0 | 76 |
| 4 | 0.0532 | 0 | 19 | 4 | 0.0532 | 0 | 19 |
| 5 | 0.0561 | 0 | 19 | 5 | 0.0561 | 0 | 19 |
| 6 | 0.0589 | 0 | 19 | 6 | 0.0589 | 0 | 19 |
| 7 | 0.0611 | 0 | 19 | 7 | 0.0475 | 0 | 58 |
| 8 | 0.0601 | 0 | 27 | 8 | 0.0601 | 0 | 27 |
| 9 | 0.0572 | 0 | 19 | 9 | 0.0572 | 0 | 19 |
| 10 | 0.0484 | 0.0002 | 76 | 10 | 0.0391 | 0 | 94 |

[‡] starting point

[†] maximum constraint violation

Table 3.9: SLSQP Optimization Results

| Avionics box height formulation | | | | Maximum lift formulation | | | |
|---------------------------------|---------------|----------------|-------------|--------------------------|---------------|----------------|------------|
| sp [‡] | f | V [†] | evals. | sp [‡] | f | V [†] | evals. |
| 1 | 0.0396 | 0 | 1825 | 1 | 0.0334 | 0 | 585 |
| 2 | 0.0402 | 0 | 3485 | 2 | 0.0371 | 0 | 510 |
| 3 | 0.0391 | 0 | 1905 | 3 | 0.0299 | 0 | 640 |
| 4 | 0.0495 | 0 | 640 | 4 | 0.0354 | 0 | 260 |
| 5 | 0.0376 | 0 | 4160 | 5 | 0.0392 | 0 | 390 |
| 6 | 0.0404 | 0 | 4550 | 6 | 0.0281 | 0 | 445 |
| 7 | 0.0511 | 0 | 900 | 7 | 0.0332 | 0 | 380 |
| 8 | 0.0571 | 0 | 120 | 8 | 0.0571 | 0 | 120 |
| 9 | 0.0470 | 0 | 1055 | 9 | 0.0312 | 0 | 390 |
| 10 | 0.0374 | 0 | 3235 | 10 | 0.0405 | 0 | 185 |

[‡] starting point

[†] maximum constraint violation

Table 3.10: LMM-BFGS Optimization Results

| Avionics box height formulation | | | | Maximum lift formulation | | | |
|---------------------------------|---------------|----------------|-------------|--------------------------|---------------|----------------|-------------|
| sp [‡] | f | V [†] | evals. | sp [‡] | f | V [†] | evals. |
| 1 | 0.0402 | 0 | 2170 | 1 | 0.0300 | 0 | 315 |
| 2 | 0.0377 | 0.0008 | 6225 | 2 | 0.0266 | 0 | 1020 |
| 3 | 0.0376 | 0 | 6570 | 3 | 0.0356 | 0 | 325 |
| 4 | 0.0482 | 0 | 1150 | 4 | 0.0302 | 0 | 370 |
| 5 | 0.0372 | 0 | 3670 | 5 | 0.0423 | 0 | 195 |
| 6 | 0.0383 | 0 | 3325 | 6 | 0.0258 | 0 | 510 |
| 7 | 0.0491 | 0 | 770 | 7 | 0.0291 | 0 | 455 |
| 8 | 0.0571 | 0 | 120 | 8 | 0.0571 | 0 | 120 |
| 9 | 0.0458 | 0 | 2020 | 9 | 0.0313 | 0 | 390 |
| 10 | 0.0388 | 0 | 1030 | 10 | 0.0249 | 0 | 1215 |

[‡] starting point

[†] maximum constraint violation

Table 3.11: LMM-CGPR Optimization Results

| Avionics box height formulation | | | | Maximum lift formulation | | | |
|---------------------------------|---------------|----------------|-------------|--------------------------|---------------|----------------|-------------|
| sp [‡] | f | V [†] | evals. | sp [‡] | f | V [†] | evals. |
| 1 | 0.0403 | 0.0018 | 9639 | 1 | 0.0296 | 0 | 8310 |
| 2 | 0.0385 | 0.0023 | 7696 | 2 | 0.0283 | 0 | 8346 |
| 3 | 0.0394 | 0.0018 | 9605 | 3 | 0.0260 | 0 | 9019 |
| 4 | 0.0433 | 0.0023 | 5899 | 4 | 0.0303 | 0 | 7213 |
| 5 | 0.0405 | 0.0026 | 9733 | 5 | 0.0291 | 0 | 10045 |
| 6 | 0.0401 | 0.0018 | 9595 | 6 | 0.0247 | 0 | 9638 |
| 7 | 0.0538 | 0.0018 | 4506 | 7 | 0.0304 | 0 | 5800 |
| 8 | 0.0570 | 0 | 1944 | 8 | 0.0570 | 0 | 1944 |
| 9 | 0.0520 | 0 | 2562 | 9 | 0.0520 | 0 | 2562 |
| 10 | 0.0376 | 0.0022 | 9017 | 10 | 0.0275 | 0 | 9762 |

[‡] starting point

[†] maximum constraint violation

Table 3.12: SQP Optimization Results

in this region of the design space is not positive-definite. The result is that the SQP algorithm resorted to its fail-safe and used steepest-descent on the penalty function to determine the search directions. Table 3.12 shows the performance of the SQP method from the starting points.

The SQP methods achieved better results for the maximum lift formulation. The QP problems could be solved in the regions of the design space which had a lower avionics box height, allowing the SQP method to perform better. Although more expensive than the other gradient-based methods, it achieved the lowest function value of 0.0247 after 9638 function evaluations.

The LMM leap frog constrained optimizer obtained the lowest function value of the gradient-based optimizers for the avionics box height formulation but at a high cost, obtaining an objective function value of 0.0365 after 25 292 function evaluations. The LMM-LFOP results, captured in Table 3.13, show the cost of this method where the average number of function evaluations is over 10 000.

This implementation of the LFOPC method struggles to determine a valid solution for the avionics box height formulation. The method often fails at the second phase of the method. The second phase of the method assumes that a higher penalty factor would successfully steer the particle to a more valid region of the design space. In the avionics box height formulation the particle often did not manage to navigate to a more valid space. The LFOPC results are shown in Table 3.14.

The LFOP-based method's maximum lift optimization runs were terminated before completion due to their long running duration. The LFOPC and LMM-LFOP methods are exceptionally slow as the maximum lift formulation steers the particle into undefined

| sp [‡] | f | V [†] | evals. |
|-----------------|---------------|----------------|--------------|
| 1 | 0.0367 | 0 | 13314 |
| 2 | 0.0399 | 0 | 13467 |
| 3 | 0.0369 | 0 | 17568 |
| 4 | 0.0466 | 0 | 2576 |
| 5 | 0.0370 | 0 | 8209 |
| 6 | 0.0366 | 0 | 13492 |
| 7 | 0.0452 | 0 | 3878 |
| 8 | 0.0602 | 0 | 977 |
| 9 | 0.0365 | 0 | 25292 |
| 10 | 0.0402 | 0 | 13932 |

‡ starting point

† maximum constraint violation

Table 3.13: LMM-LFOP avionics box height variant optimization results

| sp [‡] | f | V [†] | evals. |
|-----------------|---------------|----------------|-------------|
| 1 | 0.0451 | 0 | 9367 |
| 2 | 0.0444 | 0 | 6368 |
| 3 | 0.0354 | 0.0047 | 11039 |
| 4 | 0.0373 | 0.0161 | 1056 |
| 5 | 0.0356 | 0.0046 | 8937 |
| 6 | 0.0353 | 0.0033 | 5135 |
| 7 | 0.0337 | 0.0090 | 4308 |
| 8 | 0.0600 | 0 | 1906 |
| 9 | 0.0529 | 0 | 5133 |
| 10 | 0.0543 | 0 | 3713 |

‡ starting point

† maximum constraint violation

Table 3.14: LFOPC avionics box height variant optimization results

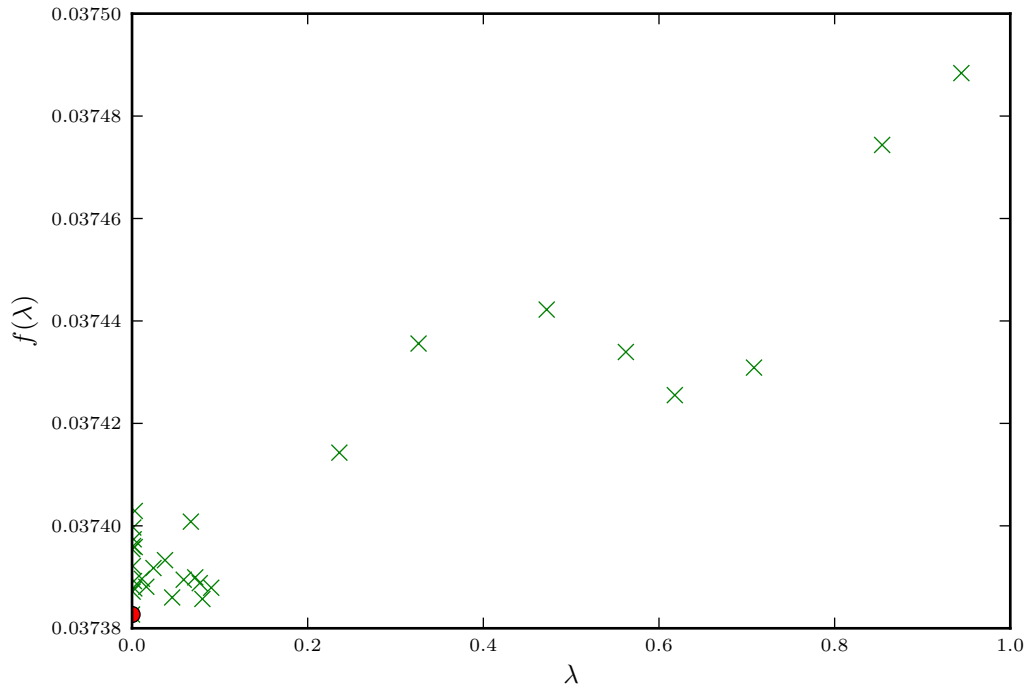
regions. Due to the exploratory nature of the algorithm, many function evaluations were spent in costly undefined regions. Evaluating a point in the undefined region takes longer than defined regions as each XFOIL setting needs to be tried in case one of them might work.

As the function minimum is approached, the gradient methods increasingly struggle to determine the correct search direction. Figure 3.9 shows the line search and gradient calculation error at an algorithm termination point. The line search shows the magnitude of the noise amplitude present in the objective-function when minima regions are approached.

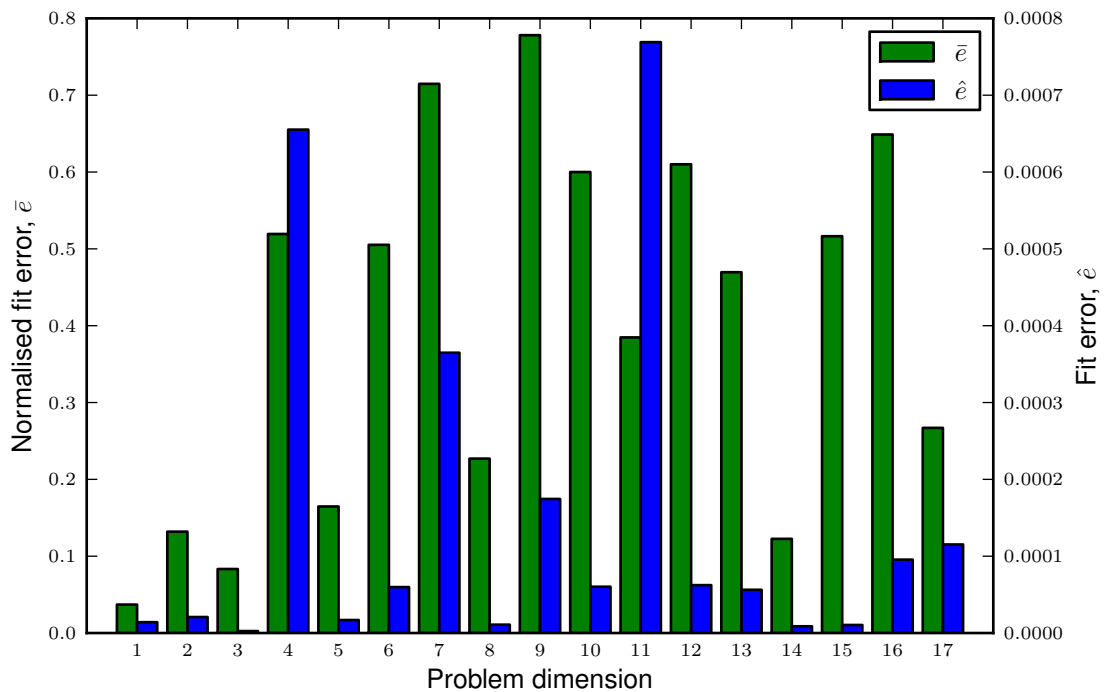
The noise present in the design space decreases the accuracy of the gradient approximations, as shown in section 3.1.3. As the optimizer approaches the minimum of the objective function, the odds of this compromised search direction leading to descent decreases.

The CSIR in their optimization work obtained a blended drag value of 0.0367 for the avionics box height formulation using the LFOPC algorithm. The gradient algorithms here achieve similar results for the avionics box height formulation and have (with limited success) been extended to handle the maximum lift formulation. The implemented gradient methods also allow for task-parallelism. This allows for parallel computing which reduces the time required to run the optimization algorithms. The best designs found by the gradient methods for the maximum lift and avionics box height formulations are shown in Figure 3.10.

The customized line search method improves the gradient-based methods final function value for avionics box height formulation but at a high number of function evaluations. However, the *a priori* formulations are not suited for gradient-based methods which assume characteristics not present and hence cannot perform as designed. The next chapter presents single objective population-based methods which are better suited to the optimization model's characteristics.

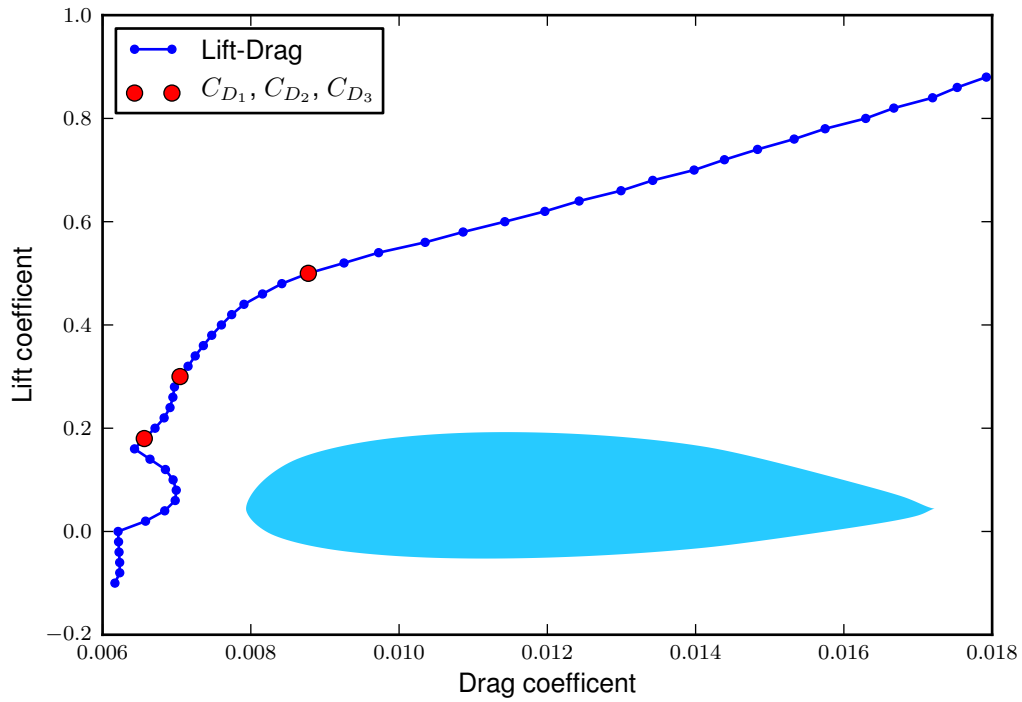


(a) line search plot

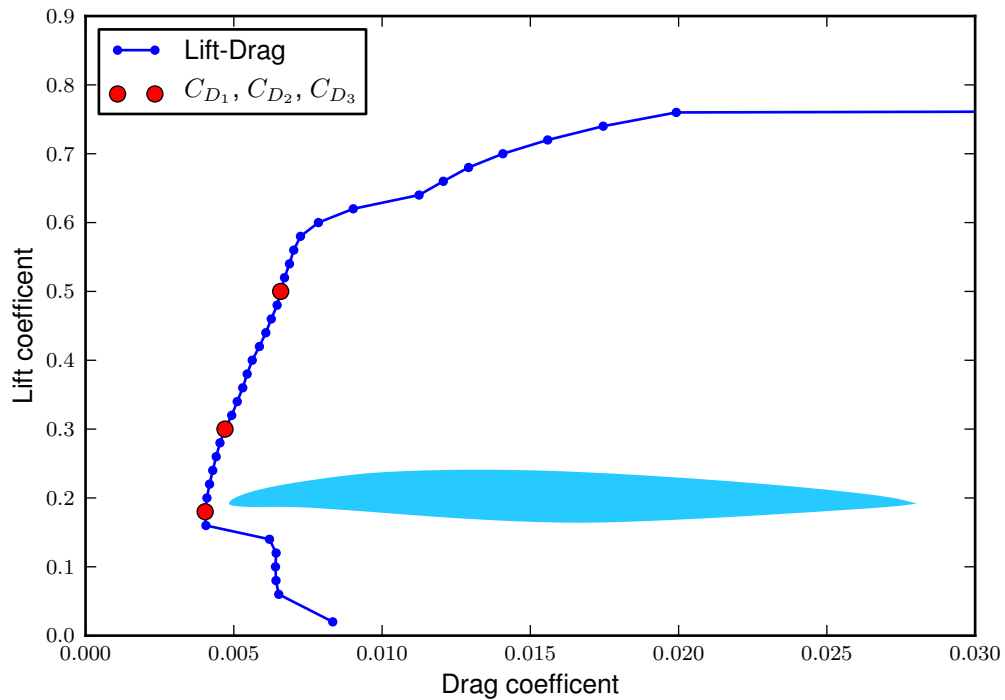


(b) gradient calculation dimension linear fit errors (3.12)

Figure 3.9: Data from the termination point of the first BFGS sub optimization of the first LMM-BFGS optimization.



(a) avionics box height formulation



(b) maximum lift formulation

Figure 3.10: Lift-Drag curves for the best design determined by the gradient-based methods.

CHAPTER 4

A PRIORI SCALARIZATION OPTIMIZATION USING POPULATION-BASED METHODS

Populations-based methods emulate group dynamics in order to search the design space and locate the problem minimum. The dynamics are influenced by other members in the population combined with stochastic effects. The methods have the favourable attributes that they do not require gradient information or smoothness.

Population-based methods' exploratory nature allows constraints to be handled easier than gradient-based methods. Gradient-methods cannot handle discontinuities in the objective space were population-based methods can. The populations-based methods only need to determine if one solution is better than another, not by how much one design is better than the other.

This chapter begins with the rules governing point selection. The Differential Evolution (DE) method is then discussed followed by a description of the Particle Swarm Optimization (PSO) method. The methods are then benchmarked on some popular population-based testing functions. The chapter concludes with the results for the airfoil *a priori* optimization using the population-based methods.

4.1 Rules governing point selection

Both the DE and PSO methods make use of the greedy selection criteria. DE uses the greedy criteria when updating population member positions, and the PSO method uses the greedy criteria when updating personal best and the global best designs.

The greedy criteria implies that when the algorithm needs to choose which design is

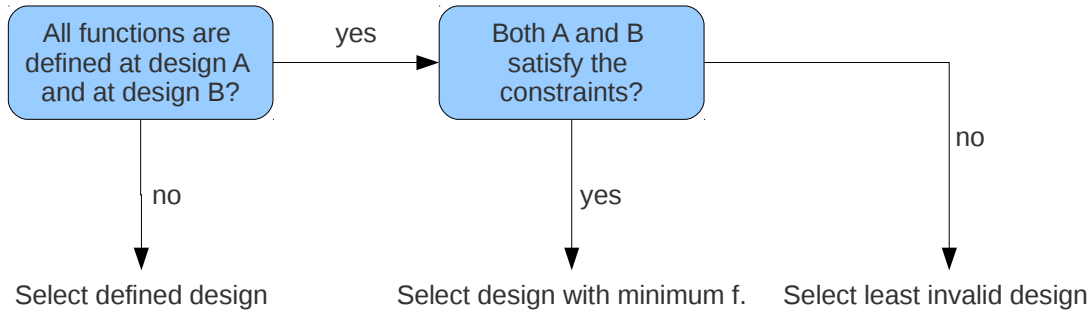


Figure 4.1: The greedy selection criteria for single objective population-based methods

better, it selects the design with the lower cost function value. For our application the greedy selection criteria is expanded to handle constraints as well as undefined regions in the design space. The rules for selecting the preferred design between two candidates are outlined in Figure 4.1.

The first criterion for comparison is to check if all the functions are defined at both points in the design space. This rule assists the optimizer to handle the undefined regions present in the design space, by giving preference to designs which are defined.

The second criterion is the validity of the design according to the optimization constraints. If both designs are valid then the design with the lowest function value will be selected. If only one design is valid it will be selected. If both of the designs are invalid the design which is more valid is selected. In this application the sum-square of the constraint violations is used to determine which design is more valid.

This decision logic can be inserted into a **design point** programming class for easy implementation. Implementing this logic in the “<” operator for the **design point** class is recommend. In the Python programming language it allows the better point C to be selected simply by $C = \min(A, B)$.

4.2 Differential Evolution

The stochastic method of DE was developed to handle non-differentiable, non-linear cost functions which are multi-modal. The implementation done here is based upon [33].

DE makes uses of a population of designs from 1 to N . At each generation i , the processes of mutation, cross-over and selection occur. The population is updated until a maximum number of iterations i_{max} , is reached or another termination criterion is satisfied.

The population’s initial positions are randomly generated inside the design space. The populating space typically takes the form of a hyper-rectangle. The unrotated hyper-rectangle is defined by its upper and lower bounds, \mathbf{b}_u and \mathbf{b}_l . The j ’th member of the

population's initial position, \mathbf{x}_0^j , is generated as follows:

$$\mathbf{x}_0^j = \mathbf{b}_l + \mathbf{r}() \otimes (\mathbf{b}_u - \mathbf{b}_l), \quad (4.1)$$

where the random function $\mathbf{r}()$ returns a vector consisting of n random elements uniformly distributed between 0 and 1, and the \otimes operator is defined as

$$\mathbf{c} = \mathbf{a} \otimes \mathbf{b} \Rightarrow \quad c_i = a_i b_i. \quad \text{for } i \in 1, 2, \dots, n. \quad (4.2)$$

The mutation process is calculated using difference vectors. The difference vectors are determined using designs from the population members. For the one difference vector mutation scheme, the mutation vector for the j^{th} member of the population \mathbf{v}_i^j is

$$\mathbf{v}_i^j = \mathbf{x}_{i-1}^{r_1} + F(\mathbf{x}_{i-1}^{r_2} - \mathbf{x}_{i-1}^{r_3}), \quad (4.3)$$

where the difference amplification F , is a real constant ranging between 0 and 2, and the population members r_1, r_2 and r_3 , are randomly chosen such that they are each unique, and that none are equal to j . The *best* variation of the mutation process replaces $\mathbf{x}_{i-1}^{r_1}$ with the population member with the best design $\mathbf{x}_{i-1}^{\text{best}}$:

$$\mathbf{v}_i^j = \mathbf{x}_{i-1}^{\text{best}} + F(\mathbf{x}_{i-1}^{r_2} - \mathbf{x}_{i-1}^{r_3}). \quad (4.4)$$

The *bin* crossover operation [33] is applied after the mutation stage. The crossover takes place between the mutation vector and the population member's current design. The crossover probability $CR \in [0, 1)$, is the same for each dimension except for one dimension which is forced to change. The index q , which is forced to crossover is selected randomly each time. The candidate vector \mathbf{u}_i^j produced from the crossover process is thus

$$\mathbf{u}_{i,k}^j = \begin{cases} \mathbf{v}_{i,k}^j & CR \text{ probability OR } k = q \\ \mathbf{x}_{i,k}^{j-1} & \text{otherwise} \end{cases} \quad \text{for } k \in 1, 2, \dots, n. \quad (4.5)$$

The final stage for each generation is selection. The greedy decision logic, shown in Figure 4.1, is used to determine if the population member is updated to its associated candidate design:

$$\mathbf{x}_i^j = \min(\mathbf{u}_i^j, \mathbf{x}_i^{j-1}). \quad (4.6)$$

The algorithm for the '/best/1/bin' (mutation $\mathbf{x}_{i-1}^{\text{best}}$ variant is used/1 difference vector in mutation /bin crossover scheme) implementation of DE is shown in Algorithm 7.

Algorithm 7 DE pseudo code

```

procedure DE( $N, F, CR, i_{max}, \mathbf{b}_l, \mathbf{b}_u$ )
  for  $j \in \{1, 2, \dots, N\}$  do                                     ▷ initialize population
     $\mathbf{x}_0^j$                                                          ▷ (4.1)
  end for
  for  $i \in \{1, 2, \dots, i_{max}\}$  do
    for  $j \in \{1, 2, \dots, N\}$  do
       $\mathbf{v}_i^j$                                                          ▷ (4.3)
       $\mathbf{u}_i^j$                                                          ▷ (4.5)
       $\mathbf{x}_i^j = \min(\mathbf{x}_{i-1}^j, \mathbf{u}_i^j)$                                ▷ Figure 4.1
    end for
  end for
end procedure
  
```

4.3 Particle Swarm

The PSO algorithm for optimization of nonlinear functions [18] is briefly presented in this section. The algorithm partly originates from a modeling algorithm used to predict the flocking behavior of birds. In the model each bird's behavior is influenced by the best location it has examined (its personal best) and by the best location the entire flock has examined (the global best).

The swarm consists of N particles, which after initialization explore the design space to determine the function minimum. The particle's initial positions are determined in the same manner as in (4.1). Each particle's velocity \mathbf{v}_j is used to update its position. The position update for a particle in the swarm at iteration i is as follows:

$$\mathbf{x}_i^j = \mathbf{x}_{i-1}^j + \mathbf{v}_i^j. \quad (4.7)$$

There are two main variations for the velocity update rules; the linear and classical velocity strategies. The classical version was implemented as it maintains population diversity better and hence searches the design space more thoroughly [37]. The classical velocity update rule is

$$\mathbf{v}_i^j = \omega \mathbf{v}_{i-1}^j + c_1 \mathbf{r}() \otimes (\mathbf{x}_{pb}^j - \mathbf{x}_{i-1}^j) + c_2 \mathbf{r}() \otimes (\mathbf{x}_{gb} - \mathbf{x}_{i-1}^j). \quad (4.8)$$

The particle is attracted to both its personal best \mathbf{x}_{pb}^j as well as the global best \mathbf{x}_{gb} , which are updated after each iteration. The attraction to the personal best as well as the global best are scaled by the personal belief constant c_1 and global belief constant c_2 . c_1 and c_2 are user specified parameters which normally range between 0 and 2.

The inertia factor ω is used to remove energy from the swarm, controlling its collapse rate. Low values of ω result in the swarm converging on what it believes to be the function minimum rapidly. Higher values of ω increases the exploratory behavior of the swarm.

ω is user specified with normal values ranging between 0 and 1. Also note that in this implementation the particles start from rest,

$$\mathbf{v}_0^j = \mathbf{0}. \quad (4.9)$$

The swarm continues to move as governed by the position update rule (4.7) and velocity update rule (4.8) until the iterations exceed the allowed number of iterations i_{max} . The PSO pseudo-code is shown in Algorithm 8.

Algorithm 8 PSO pseudo code

```

procedure PSO( $N, c_1, c_2, \omega, \mathbf{b}_l, \mathbf{b}_u, i_{max}$ )
  for  $j \in \{1, 2, \dots, N\}$  do                                ▷ initialize population
     $\mathbf{x}_0^j$                                                     ▷ (4.1)
     $\mathbf{v}_0^j = \mathbf{0}$ 
     $\mathbf{x}_{pb}^j = \mathbf{x}_0^j$ 
  end for
   $\mathbf{x}_{gb} = \min(\mathbf{x}_{pb}^1, \mathbf{x}_{pb}^2, \dots, \mathbf{x}_{pb}^N)$                     ▷ Figure 4.1
  for  $i \in \{1, 2, \dots, i_{max}\}$  do
    for  $j \in \{1, 2, \dots, N\}$  do
       $\mathbf{v}_i^j$                                                     ▷ (4.8)
       $\mathbf{x}_i^j$                                                     ▷ (4.7)
       $\mathbf{x}_{pb}^j = \min(\mathbf{x}_{pb}^j, \mathbf{x}_i^j)$                             ▷ Figure 4.1
    end for
     $\mathbf{x}_{gb} = \min(\mathbf{x}_{pb}^1, \mathbf{x}_{pb}^2, \dots, \mathbf{x}_{pb}^N)$             ▷ Figure 4.1
  end for
end procedure

```

4.4 Testing

The testing functions serve both for checking the algorithm implementations and to investigate which settings to use on the airfoil *a priori*. The test functions which are unconstrained are chosen from [37] with the exception that the extended Rosenbrock function is taken from [34]. The test functions are:

- The extended Rosenbrock function:

$$t_1(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2), \quad (4.10)$$

with populating bounds of ± 30 for each dimension.

- The Quadric function:

$$t_2(x_1, x_2, \dots, x_n) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2, \quad (4.11)$$

with populating bounds of ± 100 for each dimension.

- The Rastrigin function:

$$t_3(x_1, x_2, \dots, x_n) = \sum_{i=1}^n (x_i^2 - 10 \cos(2\pi x_i) + 10), \quad (4.12)$$

with populating bounds of ± 5.12 for each dimension.

- The Griewank function:

$$t_4(x_1, x_2, \dots, x_n) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad (4.13)$$

with populating bounds of ± 600 for every dimension.

The minimum for the test functions occur at $[0, 0, \dots, 0]^T$ except for the extended Rosenbrock function whose minimum is at $[1, 1, \dots, 1]^T$. The population size for testing is 50, which is the same size that is used for the airfoil *a priori* optimization. The maximum number of allowed function evaluations is 200 000. A choice of 2 000 or 5 000 for the maximum number of allowed function evaluations may be more approximate as this is the number of function evaluations that will be used for the airfoil optimization, but 200 000 is used as it a standard testing amount allowing for comparison again other results.

For each testing problem one parameter is varied to determine its effect on the algorithm performance. The cross-over probability CR is swept for the DE algorithm and the inertia factor ω is swept for PSO. The other algorithm setting remain constant, with $F = 0.3^1$ for the DE method and $c_1 = 1$ and $c_2 = 1$ for the PSO method.

The algorithm's performance is averaged over 100 runs for each setting used. This is done as the algorithm returns a different result each time it is run. The results from these stochastic algorithms form a probability distribution. The average gives an approximation of the mean of this distribution.

The results indicate that the PSO algorithm performs best for ω values greater than 0.6. Figure 4.2 shows the results of the PSO algorithm on the testing functions. It is likely due to the unconventional choice of c_1 , c_2 that the best ω values are larger than 0.7298 which is a value often used in PSO literature such as [10].

¹Initially a F value of 0.5 was chosen but reducing it to 0.3 produced better results for the airfoil single-objective optimization.

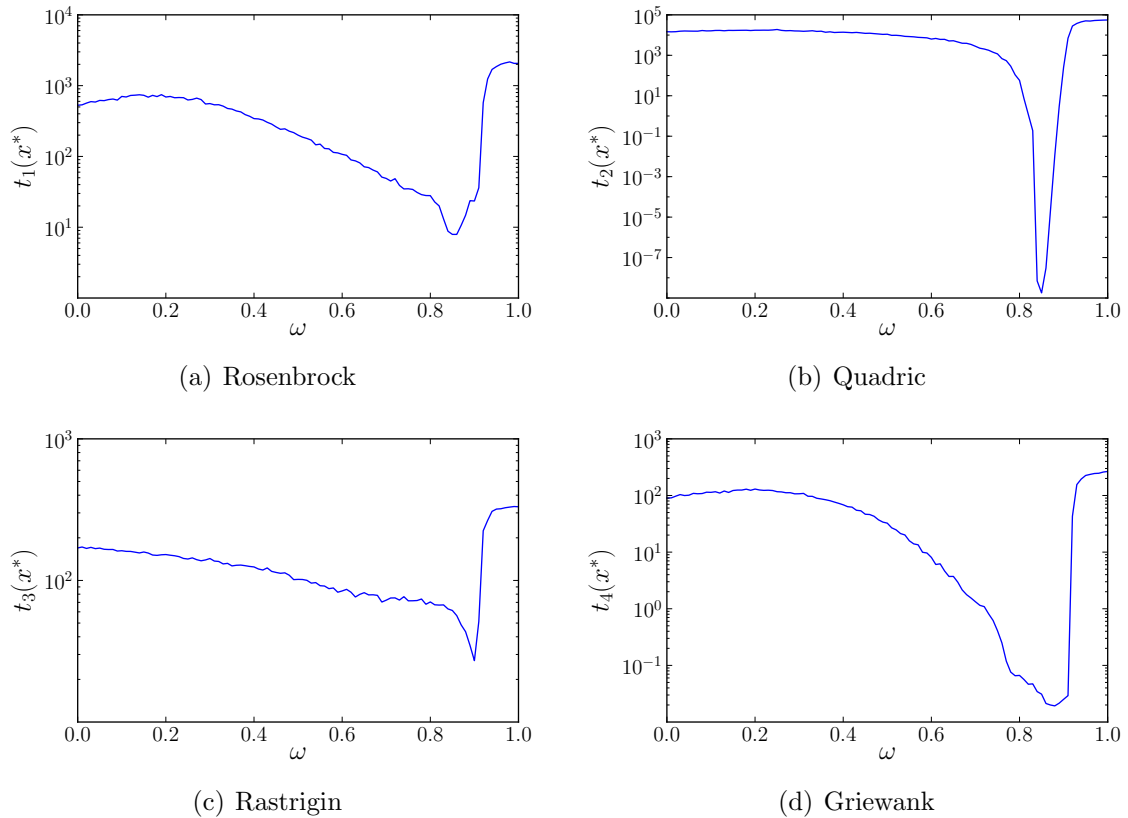


Figure 4.2: PSO performance on the testing functions for various ω .

The DE algorithm performs best on lower values of CR . Figure 4.3 shows the DE results. This is expected as the testing functions are either loosely coupled or uncoupled. For the airfoil optimization where the problem is coupled a higher CR is used.

The algorithms can handle constrained optimization problems successfully solving many of the problem from [19]. The methods are however more expensive than the gradient-based methods for these smooth differentiable functions. This result is important as it confirms that the algorithm's selection can handle constraints. Appendix D presents the results from these tests.

4.5 Airfoil optimization

The population-based method's results on the *a priori* airfoil formulations are presented in this section. Each setting tried was run multiple times to estimate how much variation was present. The airfoil single objective population-based optimization formulations, reproduced for convenience are:

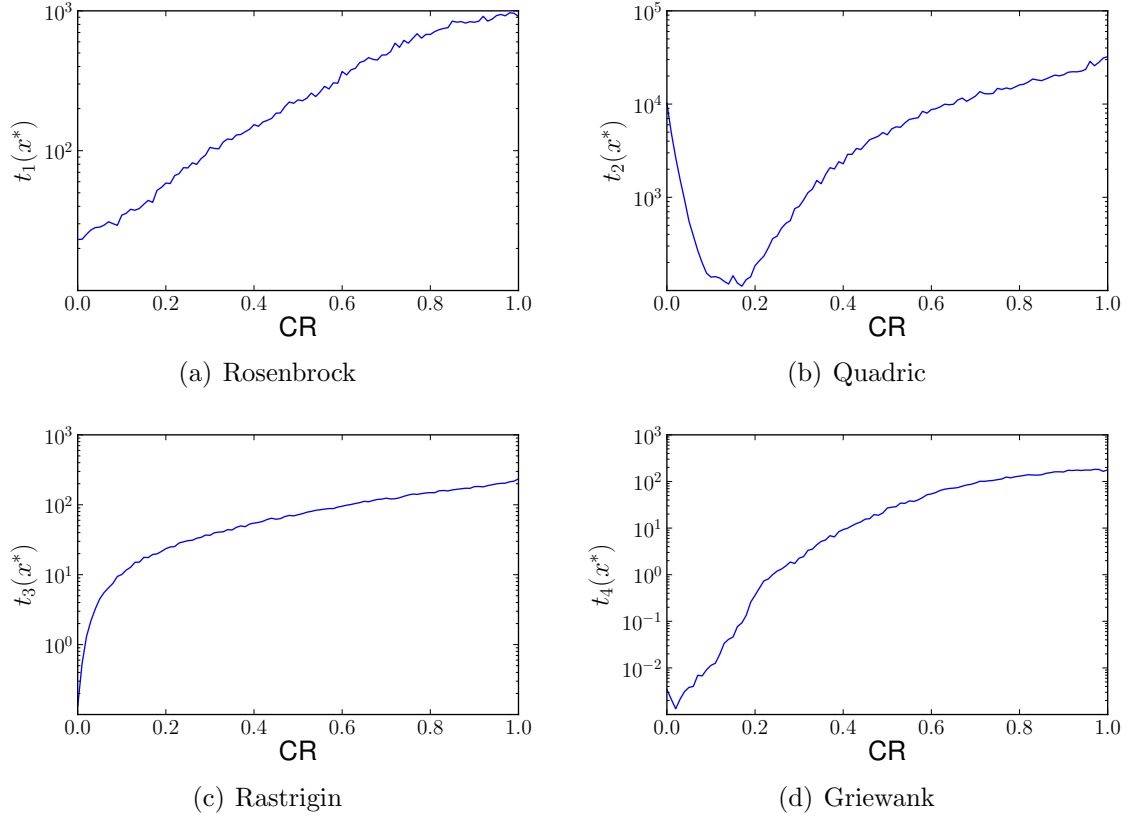


Figure 4.3: DE performance on the testing functions for various CR.

1. Avionics box height variant:

$$f(\mathbf{x}) = 3C_{D_1}(\mathbf{x}) + C_{D_2}(\mathbf{x}) + C_{D_3}(\mathbf{x}) \quad (4.14)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \max\text{Lift}(\mathbf{x}) \\ 73\text{mm} - B_h(\mathbf{x}) \end{bmatrix} \quad (4.15)$$

$$\mathbf{B}_l = [0, 0, \dots, 0] \quad (4.16)$$

$$\mathbf{B}_u = [1, 1, \dots, 1] \quad (4.17)$$

2. Maximum lift variant:

$$f(\mathbf{x}) = 3C_{D_1}(\mathbf{x}) + C_{D_2}(\mathbf{x}) + C_{D_3}(\mathbf{x}) \quad (4.18)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \max\text{Lift}(\mathbf{x}) \\ 10\text{mm} - B_h(\mathbf{x}) \end{bmatrix} \quad (4.19)$$

$$\mathbf{B}_l = [0, 0, \dots, 0] \quad (4.20)$$

$$\mathbf{B}_u = [1, 1, \dots, 1] \quad (4.21)$$

The function evaluations are limited to 5000 for the avionics box height formulation and 2000 for the maximum lift formulation. The function evaluations for the maximum

| settings | | after 1000 evals. | | after 5000 evals. | |
|----------|-----|-------------------|-------------|-------------------|-------------|
| CR | F | f_{best} | V^\dagger | f_{best} | V^\dagger |
| 0.60 | 0.3 | 0.03991 | 0 | 0.03612 | 0 |
| 0.60 | 0.3 | 0.03985 | 0 | 0.03596 | 0 |
| 0.60 | 0.3 | 0.03736 | 0 | 0.03625 | 0 |
| 0.70 | 0.3 | 0.03750 | 0 | 0.03621 | 0 |
| 0.70 | 0.3 | 0.04017 | 0 | 0.03614 | 0 |
| 0.70 | 0.3 | 0.03990 | 0 | 0.03636 | 0 |
| 0.80 | 0.3 | 0.03790 | 0 | 0.03630 | 0 |
| 0.80 | 0.3 | 0.03780 | 0 | 0.03632 | 0 |
| 0.80 | 0.3 | 0.04206 | 0 | 0.03659 | 0 |

[†] maximum constraint violation.

Table 4.1: DE optimization results for the avionics box height formulation.

lift formulation are limited to 2000 due to additional computation cost required to analyze airfoils where the maximum lift constraint becomes active.

The optimization results for the DE and the PSO algorithms on the avionics box height formulation are summarized in Tables 4.1 and 4.2 respectively. The best designs after 1000 function evaluations are also presented to indicate the algorithms' performances at a lower number of function evaluations.

The DE algorithm consistently returns a low objective function value for the avionics box height formulation. The worst run from the DE is only marginally worse than the best run from all the gradient methods. Its best design after 1000 evaluation is 0.03736 and after 5000 evaluations is 0.03596.

The PSO algorithm does not perform as consistently as the DE algorithm but achieved the lowest objective function value. The worst PSO run returns an airfoil with a blended drag of 0.04 and the best airfoil's blended drag is 0.03591. Figure 4.4 shows the best design determined by the population-based methods.

Tables 4.3 and 4.4 summarizes the performance for the DE and PSO algorithms for the maximum lift formulation. As in the avionics box height formulation the value after 1000 function evaluations are also presented in these tables. For reference the lowest blended drag obtained by the gradient methods is 0.0247 after 9638 function evaluations.

The majority of the DE runs achieve a blended drag coefficient of lower than 0.022 for the maximum lift optimization. The two outlier points achieved a blended drag of 0.02856 and 0.0251. The lowest blended drag achieved after 1000 evaluations is 0.0212 which is slightly larger then the minimum value of 0.0203 after 2000 evaluations.

| settings | | | after 1000 evals. | | after 5000 evals. | |
|----------|-------|-------|-------------------|-------------|-------------------|-------------|
| ω | c_1 | c_2 | f_{gb} | V^\dagger | f_{gb} | V^\dagger |
| 0.60 | 1.0 | 1.0 | 0.04269 | 0 | 0.04036 | 0 |
| 0.60 | 1.0 | 1.0 | 0.03789 | 0 | 0.03720 | 0 |
| 0.60 | 1.0 | 1.0 | 0.03897 | 0 | 0.03748 | 0 |
| 0.70 | 1.0 | 1.0 | 0.04009 | 0 | 0.03804 | 0 |
| 0.70 | 1.0 | 1.0 | 0.03782 | 0 | 0.03720 | 0 |
| 0.70 | 1.0 | 1.0 | 0.03738 | 0 | 0.03635 | 0 |
| 0.80 | 1.0 | 1.0 | 0.03779 | 0 | 0.03591 | 0 |
| 0.80 | 1.0 | 1.0 | 0.03874 | 0 | 0.03601 | 0 |
| 0.80 | 1.0 | 1.0 | 0.03796 | 0 | 0.03683 | 0 |

[†] maximum constraint violation.

Table 4.2: PSO results for the avionics box height formulation.

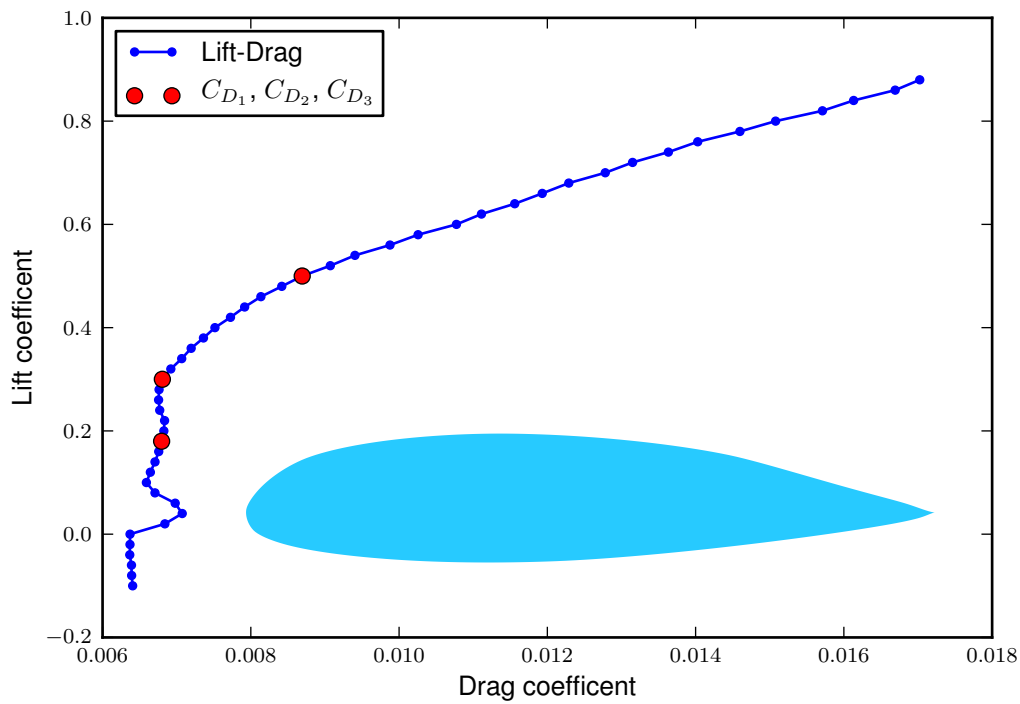


Figure 4.4: Lift-Drag curve for the best design from the PSO runs for the avionics box height formulation.

| settings | | after 1000 evals. | | after 2000 evals. | |
|----------|-----|-------------------|-------------|-------------------|-------------|
| CR | F | f_{best} | V^\dagger | f_{best} | V^\dagger |
| 0.60 | 0.3 | 0.02547 | 0 | 0.02108 | 0 |
| 0.60 | 0.3 | 0.02227 | 0 | 0.02151 | 0 |
| 0.60 | 0.3 | 0.02276 | 0 | 0.02038 | 0 |
| 0.70 | 0.3 | 0.02264 | 0 | 0.02149 | 0 |
| 0.70 | 0.3 | 0.02120 | 0 | 0.02107 | 0 |
| 0.70 | 0.3 | 0.02542 | 0 | 0.02131 | 0 |
| 0.80 | 0.3 | 0.02926 | 0 | 0.02856 | 0 |
| 0.80 | 0.3 | 0.02534 | 0 | 0.02518 | 0 |
| 0.80 | 0.3 | 0.02391 | 0 | 0.02153 | 0 |

[†] maximum constraint violation.

Table 4.3: DE optimization results for maximum lift formulation.

The PSO method obtains the best design for the maximum lift formulation. As in the avionics box height formulation, there appears to be large variation for results generated using the same settings. The design that corresponds to the best blended drag value of 0.01917 is shown in Figure 4.5.

Figure 4.6 shows a shape comparison between the lowest blended drag designs of the gradient method and the population method. The population based design which has the lower drag, has a rounder nose and a sharper trailing edge.

Examining the results from Chapters 3 and 4, the following statements are made regarding the *a priori* airfoil optimization formulations:

- The population-based methods were able to handle the maximum lift formulation. The poor quality of the maximum lift constraint inhibited the performance of the gradient-based and surrogate methods.
- The population-based methods performed better than the modified gradient-based methods obtaining better designs and using less function evaluations. This statement does not hold for the normal, unaltered versions of the modified gradient algorithms which use far less function evaluations than their modified counterparts, which over-sample the design space as to produce a better ultimate answer.
- For the avionics box height formulation the population based methods and the surrogate methods both performed well. Note that although the surrogate methods produced worse designs they did it using less function evaluations.

The *a priori* optimization results are based upon a simplified formulation with blended cost functions. The next chapter presents the Pareto-optimal multi-objective optimiza-

| settings | | | after 1000 evals. | | after 2000 evals. | |
|----------|-------|-------|-------------------|-------------|----------------------|-------------|
| ω | c_1 | c_2 | f_{gb} | V^\dagger | f_{gb} | V^\dagger |
| 0.60 | 1.0 | 1.0 | 0.02675 | 0 | 0.02643 | 0 |
| 0.60 | 1.0 | 1.0 | 0.02504 | 0 | 0.02379 | 0 |
| 0.60 | 1.0 | 1.0 | 0.02141 | 0 | 0.02075 | 0 |
| 0.70 | 1.0 | 1.0 | 0.02240 | 0 | 0.02139 | 0 |
| 0.70 | 1.0 | 1.0 | 0.02537 | 0 | 0.02264 | 0 |
| 0.70 | 1.0 | 1.0 | 0.02152 | 0 | 0.01581 [‡] | 0 |
| 0.80 | 1.0 | 1.0 | 0.02755 | 0 | 0.02214 | 0 |
| 0.80 | 1.0 | 1.0 | 0.02241 | 0 | 0.01917 | 0 |
| 0.80 | 1.0 | 1.0 | 0.02249 | 0 | 0.02151 | 0 |

[†] maximum constraint violation.

[‡] this result is ignored as it is a numerical artefact not corresponding to the physical solution.

Table 4.4: PSO results for maximum lift formulation.

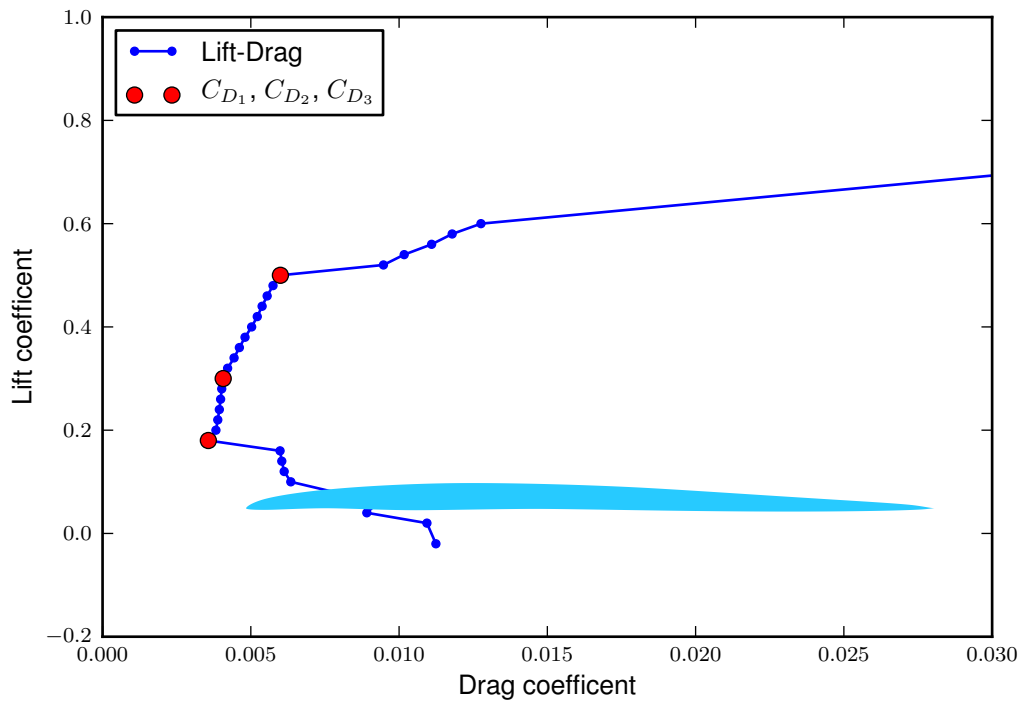


Figure 4.5: Lift-Drage curve for the best design from the PSO runs for the maximum lift formulation.

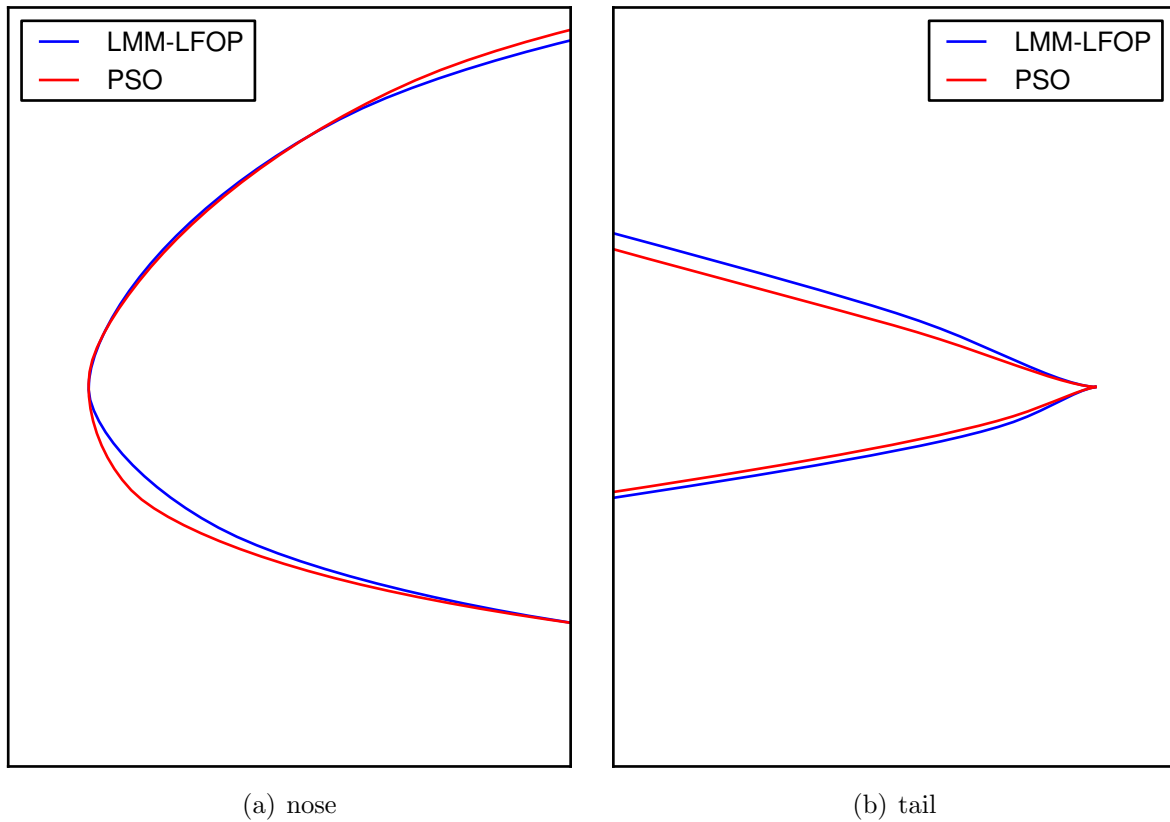


Figure 4.6: Airfoil shape comparison between LMM-LFOP and PSO best designs for the avionics box height formulation.

tion which does not require such simplifications. Pareto-optimal multi-objective approaches provide the user with a more in-depth understanding of the problem's characteristics and the effect different trade-off choices will have during the design process.

CHAPTER 5

PARETO-OPTIMAL MULTI-OBJECTIVE OPTIMIZATION

Implementing Pareto-optimal multiple objective optimization allows for a more detailed analysis. When problems with multiple objectives are reduced to a single objective formulation, the user is forced to make simplifications. These simplifications can be done by posing some of the objectives as constraints, or by giving weights to each objective and summing them up. A detailed survey of multi-objective optimization methods can be found in [24].

In this application of airfoil optimization, the *a priori* formulations are created by blending the objectives. Since the UAV is likely to spend the majority of its flight time in a cruise mode, the cruise drag coefficient (C_{D_1}) is given three times the importance of the loiter and high-speed dash drag coefficients, (C_{D_2}) and (C_{D_3}):

$$f(\mathbf{x}) = 3C_{D_1}(\mathbf{x}) + C_{D_2}(\mathbf{x}) + C_{D_3}(\mathbf{x}). \quad (5.1)$$

The choice of weights is difficult to justify, requiring prior knowledge about the influence of each objective. The following questions arise regarding the weight set of $\{3, 1, 1\}$ used in (5.1):

- How sensitive is the solution to the weights? Would weights of $\{2, 1, 1\}$ yield a significantly different solution?
- How would an optimized design focusing on loiter or high-speed dash performance differ from the optimized design of (5.1)?
- To gain a small increase in loiter performance how much will the drag of the other objectives increase?

Performing a multi-objective analysis and determining the Pareto front allows for questions of this nature to be answered.

Direct multi-objective population-based algorithms have been receiving significant attention since the mid-1980's when the VEGA Algorithm was introduced [11]. The methods are popular amongst engineering disciplines where objective trade-offs are common.

Direct multi-objective techniques have been successfully applied to airfoil shape optimization. One example is a multi-disciplinary shape optimization of aerodynamics and electromagnetics using genetic algorithms [23]. The paper presents work on a multi-objective airfoil optimization, where the objectives are to minimize drag and to minimize radar cross-section, subject to a maximum lift constraint. A genetic algorithm together with parallel computing is used for the multi-objective optimization. Another more recent example, is where a jet aircraft wing has been optimized for three conflicting objectives [9].

In this chapter multi-objective definitions and concepts are first discussed. Then the MOPSO, MOSADE, and EPO algorithms which are used for the airfoil multi-objective optimizations are presented. The chapter concludes with the multi-objective results obtained for various airfoil multi-objective formulations.

5.1 Definitions

The multi-objective function \mathbf{F} , is a vector of k objective functions. Each objective function depends on n design variables.

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \dots \\ f_k(\mathbf{x}) \end{bmatrix} \quad \mathbf{x} \in \mathfrak{R}^n, \quad (5.2)$$

subject to equality constraints and inequality constraints, $\mathbf{h}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$, respectively. For practical consideration a tolerance, ϵ_h , is applied to the equality constraint. The set of all valid designs \mathfrak{X} is defined as

$$\mathfrak{X} = \left\{ \mathbf{x} \text{ if } \text{all}(\mathbf{g}(\mathbf{x}) \leq \mathbf{0}) \text{ and } \text{all}(-\epsilon_h \leq \mathbf{h}(\mathbf{x}) \leq \epsilon_h) \forall \mathbf{x} \in \mathfrak{R}^n \right\}, \quad (5.3)$$

where the “all” operator returns true if all the elements in a vector pass the logical criterion.

A design \mathbf{y} dominates another design \mathbf{z} if it is better or equal, for every objective. Another requirement is that at least one of the objectives of \mathbf{y} is less than the corresponding objective of \mathbf{z} . Since in this text the aim is to minimize all the objectives, the

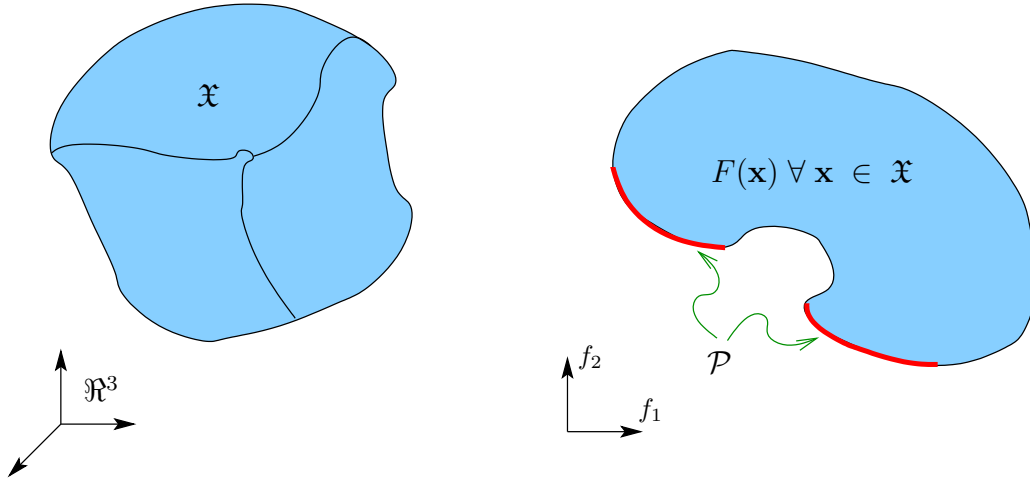


Figure 5.1: Illustrating the concept of feasible design space \mathfrak{X} , objective space and the Pareto front \mathcal{P} .

dominance logical operator is defined as

$$\mathbf{y} \prec \mathbf{z} \text{ if all } (\mathbf{F}(\mathbf{y}) \leq \mathbf{F}(\mathbf{z})) \text{ and any } (\mathbf{F}(\mathbf{y}) < \mathbf{F}(\mathbf{z})), \quad (5.4)$$

where the “any” operator returns true if any of the elements in a vector passes the logical criterion.

If \mathbf{y} does not dominate \mathbf{z} ($\mathbf{y} \not\prec \mathbf{z}$), it does NOT imply that $\mathbf{z} \prec \mathbf{y}$. It often occurs that \mathbf{y} is better in some objectives but worse in others when compared to \mathbf{z} , with neither design dominating the other. A design is non-dominated if no other designs exist (in the valid design space) that dominates it. A non-dominated design χ is defined as:

$$\chi \not\prec \mathbf{x} \text{ for all } \mathbf{x} \in \mathfrak{X}. \quad (5.5)$$

The Pareto front \mathcal{P} is the set of all non-dominated designs. For many applications there are infinite points on the Pareto front. The goal of a multi-objective optimizer is to determine \mathcal{P} i.e. to find the Pareto-optimal non-dominated designs. The concepts of design space, objective space and the Pareto front are illustrated in Figure 5.1.

5.2 MOPSO

The multi-objective particle swarm optimization (MOPSO) algorithm makes use of an external archive or repository and a mutation operator. The implementation here is based upon [12].

The repository serves both to store the Pareto front and as a reference for the optimization. The repository, also commonly referred to as the archive, stores the set of non-dominated designs (X_r) discovered by the algorithm. Each function evaluation

which is valid, is passed to the repository for inspection. Should the candidate design \mathbf{x}_c not be dominated by any design in the repository it is added, otherwise it is discarded. Also, if any designs are dominated by \mathbf{x}_c then they are discarded from the repository. These mechanics ensure that only non-dominated solutions are added to the repository. Formally,

$$\text{add } \mathbf{x}_c \text{ to } X_r \text{ if not any } \mathbf{x} \prec \mathbf{x}_c \forall \mathbf{x} \in X_r, \quad (5.6)$$

$$\text{discard from } X_r \text{ all } \mathbf{x} \text{ where } \mathbf{x}_c \prec \mathbf{x}. \quad (5.7)$$

The MOPSO repository makes use of an adaptive grid to space the non-dominated designs. The grid consists of d divisions along each objective dimension. The grid's upper and lower boundaries, \mathbf{r}_u and \mathbf{r}_l , are determined to fit the smallest possible unrotated hyper-rectangle around X_r . The grid boundaries are updated every time a solution is added which falls outside the current bounds.

Each design in X_r has grid indexes assigned to it according to its objective function values. The grid location \mathbf{z} for a design $\mathbf{x} \in X_r$ as a function of the grid boundaries is

$$z_j = \left\lfloor \frac{F_j(\mathbf{x}) - r_{l,j}}{r_{u,j} - r_{l,j}} d \right\rfloor \quad j \in 1, 2, \dots, k. \quad (5.8)$$

The repository is limited to a maximum size of r_m . Should the size of X_r exceed r_m the oldest design from the most densely populated grid division r_{dp} is discarded. The user's selection of r_m should be linked to d so that the grid is not too sparsely populated or over-populated.

The pseudo code for adding a point to the repository is given in Algorithm 9.

Algorithm 9 MOPSO pseudo code for adding a point to the repository

```

procedure REP INSPECT( $\mathbf{x}_c$ )
  if  $\mathbf{x}_c$  valid then                                     ▷ does not violate any constraints
    update  $X_r$                                            ▷ (5.6) and (5.7)
    if  $\mathbf{x}_c$  added and  $\mathbf{x}_c$  outside bounds then
       $\mathbf{r}_u, \mathbf{r}_l$ 
      re-calculated  $X_r$  grid locations
    end if
    if size( $X_r$ ) >  $r_m$  then
      eliminate oldest non-dominated solution from  $r_{dp}$ 
    end if
  end if
end procedure

```

The particle swarm, consisting of N particles, is randomly spawned inside a hyper-rectangle bound between \mathbf{b}_l and \mathbf{b}_u . This is done in the same manner as in the PSO and DE algorithms (4.1). After initialization each particle's design is passed to the repository

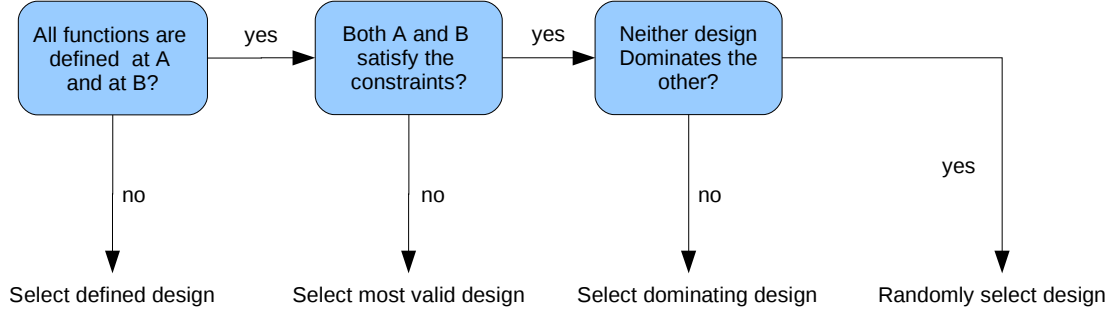


Figure 5.2: MOPSO selecting between two designs, A and B

for inspection. At each iteration i , position \mathbf{x}_i^j , and the velocity \mathbf{v}_i^j , of the j 'th member of the swarm's population are updated.

Each particle is influenced by that particle's personal best design \mathbf{x}_{pb}^j , and by $\mathbf{x}_h()$ which is a non-dominated design selected from X_r . The MOPSO velocity and position update-rules are

$$\mathbf{v}_i^j = \omega \mathbf{v}_{i-1}^j + r_1() (\mathbf{x}_{pb}^j - \mathbf{x}_{i-1}^j) + r_2() (\mathbf{x}_h() - \mathbf{x}_{i-1}^j) \quad (5.9)$$

$$\mathbf{x}_i^j = \mathbf{x}_{i-1}^j + \mathbf{v}_i^j. \quad (5.10)$$

The velocity update rule is linear with $r_1()$ and $r_2()$ generating random scalars bound between 0 and 1 generated using a uniform probability density. $\mathbf{x}_h()$ is selected from X_r using roulette-wheel selection, with the designs in X_r whom solely occupy a repository grid division given 10 times the likelihood of being selected.

The MOPSO criterion for selection of \mathbf{x}_{pb}^j is the design which dominates the other, unless neither dominates the other, in which case the design is selected randomly. The implemented rules for updating \mathbf{x}_{pb}^j are shown in Figure 5.2.

The MOPSO algorithm is designed such that the particles are boxed inside the populating hyper-rectangle. The boxing occurs after each position and velocity update. For each dimension $m \in 1, 2, \dots, n$, the position components $x_{i,m}^j$ and velocity components $v_{i,m}^j$ of every particle are confined as follows:

$$v_{i,m}^j = \begin{cases} v_{i,m}^j & \text{if } b_{l,m} < x_{i,m}^j < b_{u,m} \\ -v_{i,m}^j & \text{otherwise,} \end{cases} \quad (5.11)$$

$$x_{i,m}^j = \begin{cases} b_{l,m} & \text{if } x_{i,m}^j < b_{l,m} \\ b_{u,m} & \text{if } b_{u,m} < x_{i,m}^j \\ x_{i,m}^j & \text{otherwise.} \end{cases} \quad (5.12)$$

A mutation operator is used to increase the diversity of the swarm. The likelihood

of mutation decreases as i approaches the maximum number of iterations i_{max} . The likelihood of mutation taking place p_m , as a function of the mutation rate η is

$$p_m = (1 - i/i_{max})^{5/\eta}. \quad (5.13)$$

When mutation takes place it affects a randomly selected dimension q . A uniform-density random number generator function $r_u(b_l, b_u)$ (bound between b_l and b_u) is used to assign a new value for dimension q according to

$$x_{i,q}^j = r_u \left(\max(b_{l,q}, x_{i,q}^j - (b_{u,q} - b_{l,q})(1 - i/i_{max})^{5/\eta}), \min(b_{u,q}, x_{i,q}^j + (b_{u,q} - b_{l,q})(1 - i/i_{max})^{5/\eta}) \right). \quad (5.14)$$

The pseudo-code for MOPSO is give in Algorithm 10.

Algorithm 10 MOPSO pseudo code

```

procedure MOPSO( $N, \omega, \eta, i_{max}, d, \mathbf{b}_l, \mathbf{b}_u$ )
  for  $j \in \{1, 2, \dots, N\}$  do                                     ▷ initialize population
     $\mathbf{x}_0^j$                                                          ▷ (4.1)
     $\mathbf{v}_0^j = \mathbf{0}$ 
     $\mathbf{x}_{pb}^j = \mathbf{x}_0^j$ 
    Rep Inspect  $\mathbf{x}_0^j$                                              ▷ Algorithm 9
  end for
  for  $i \in \{1, 2, \dots, i_{max}\}$  do                               ▷ main loop
    for  $j \in \{1, 2, \dots, N\}$  do
       $\mathbf{v}_i^j$                                                          ▷ (5.9)
       $\mathbf{x}_i^j$                                                          ▷ (5.10)
      box particle                                               ▷ (5.11) and (5.12)
      mutation                                                  ▷ (5.13) and (5.14)
       $\mathbf{x}_{pb}^j = \min(\mathbf{x}_{pb}^j, \mathbf{x}_i^j)$                              ▷ Figure 5.2
      Rep Inspect  $\mathbf{x}_i^j$                                            ▷ Algorithm 9
    end for
  end for
end procedure

```

5.3 MOSADE

The multi-objective self-adaptive differential evolution method (MOSADE) makes use of an elitist archive and adaptive parameters to solve a multi-objective problem. The archive (here after referred to as the repository) uses a crowding entropy-based diversity measure to select which non-dominated solutions are given preference. The DE difference factor F , and crossover rate CR are not constant as in normal DE implementations, but are designed to adapt to the problem. The algorithm is implemented as described in [36].

Each design in the repository's non-dominated design set X_r , has a crowding entropy associated with it. The crowding entropy is used to determine which designs in X_r are rejected when the repository exceeds its maximum size r_m , and the crowding entropy also aids in design selection.

The first step in determining a designs crowding entropy is to create a sorted list ϕ^m of all the design components in the m 'th objective function dimension

$$\phi^m = \text{sorted } \{F_m(\mathbf{x})\} \text{ for } \mathbf{x} \in X_e \quad m \in \{1, 2, \dots, k\}, \quad (5.15)$$

where the design set X_e is defined as

$$X_e = \begin{cases} X_r & \text{if } \mathbf{x}_i \in X_r \\ X_r + \{\mathbf{x}_i\} & \text{otherwise.} \end{cases} \quad (5.16)$$

X_e is used to compute the crowding entropy for the design \mathbf{x}_i .

The crowding component c_m for \mathbf{x}_i is then calculated using the entropy function shown in Figure 5.3. The crowding entropy for \mathbf{x}_i is

$$c_m(\mathbf{x}_i) = (\phi_p^m - \phi_{p-1}^m) \log_2 \left(\frac{\phi_p^m - \phi_{p-1}^m}{\phi_{p+1}^m - \phi_{p-1}^m} \right) + (\phi_{p+1}^m - \phi_p^m) \log_2 \left(\frac{\phi_{p+1}^m - \phi_p^m}{\phi_{p+1}^m - \phi_{p-1}^m} \right) \quad (5.17)$$

where

- $\phi_p^m = F_m(\mathbf{x}_i)$,
- ϕ_{p-1}^m is the lower neighbor of ϕ_p^m in ϕ^m and
- ϕ_{p+1}^m is the upper neighbor of ϕ_p^m in ϕ^m .

Should the ϕ_p^m not have a upper or lower neighbor as it is on the boundary, a negative infinite value is given to $c_m(\mathbf{x}_i)$. In this application we shall use -1000 which is sufficient since $\phi_{p-1}^m - \phi_{p+1}^m \gg -1000$.

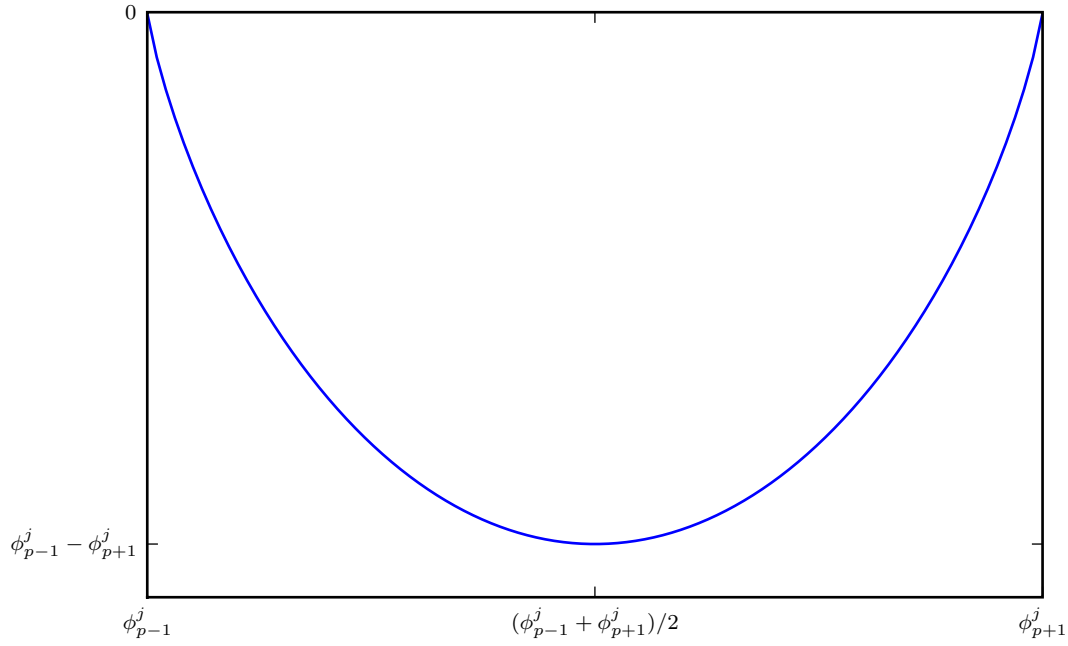


Figure 5.3: MOSADE entropy crowding function C_i^j , for various ϕ_p^j

The appeal c is obtained by adding the c_i for every dimension

$$c(\mathbf{x}_i) = - \sum_{m=1}^k c_m(\mathbf{x}_i). \quad (5.18)$$

Designs are inspected by the repository in a similar way as MOPSO (5.6) and (5.7). The repository trimming, unlike MOPSO, is not done immediately but at the end of each iteration of MOSADE. As such the size of X_r may exceed r_m by more than one. The elimination strategy is: while $\text{len}(X_r) > r_m$ eliminate the design in X_r with lowest c .

The population behavior is similar to the original DE. It makes use of the *rand/1/bin* scheme for the mutation and crossover operations. However, it has a different selection mechanism and adaptive parameters. Each population member has an associated F_j and CR_j which are generated randomly between the user specified boundaries

$$F_j = r_u(F_l, F_u) \quad (5.19)$$

$$CR_j = r_u(CR_l, CR_u). \quad (5.20)$$

If the j^{th} member failure count α_j reaches α_{max} , F_j and CR_j are regenerated. The parameters are considered to have failed if the generated candidate design is not better than the population member's current design.

The selection mechanism from [36] selects the preferred design C , by choosing between

designs A and B as follows

$$C = \begin{cases} A & \text{if } F(A) \prec F(B) \\ B & \text{if } F(B) \prec F(A) \\ A & \text{if } F(A) \not\prec F(B) \text{ and } F(B) \not\prec F(A) \text{ and } c(A) < c(B) \\ B & \text{otherwise.} \end{cases} \quad (5.21)$$

The MOSADE pseudo code is given in Algorithm 11.

Algorithm 11 MOSADE pseudo code

```

procedure MOSADE( $N, F_l, F_u, CR_l, CR_u, i_{max}, \alpha_{max}, \mathbf{b}_l, \mathbf{b}_{u, r_m}$ )
  for  $j \in \{1, 2, \dots, N\}$  do ▷ initialize population
     $\mathbf{x}_0^j$  ▷ (4.1)
    Repository Inspect  $\mathbf{c}_0^j$ 
     $F_j$  ▷ (5.19)
     $CR_j$  ▷ (5.20)
     $\alpha_j = 0$ 
  end for
  for  $i \in \{1, 2, \dots, i_{max}\}$  do ▷ main loop
    for  $j \in \{1, 2, \dots, N\}$  do
       $\mathbf{v}_i^j$  ▷ (4.3)
       $\mathbf{u}_i^j$  ▷ (4.5)
      if  $\mathbf{x}_{i-1}^j \not\prec \mathbf{u}_i^j$  then
        Repository Inspect  $\mathbf{u}_i^j$ 
      end if
      if  $\mathbf{u}_i^j < \mathbf{x}_{i-1}^j$  then ▷ (5.21)
         $\mathbf{x}_i^j = \mathbf{u}_i^j$ 
      else
         $\alpha_j = \alpha_j + 1$ 
        if  $\alpha_j = \alpha_{max}$  then
           $F_j$  ▷ (5.19)
           $CR_j$  ▷ (5.20)
           $\alpha_j = 0$ 
        end if
      end if
    end for
  end for
end procedure

```

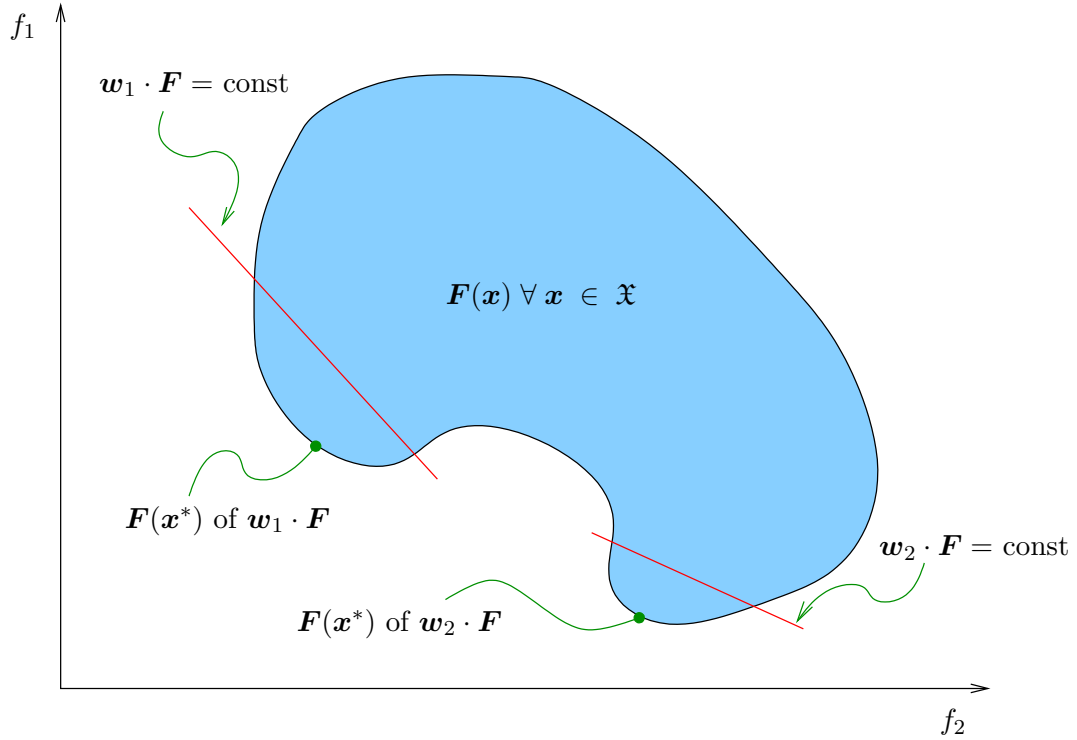


Figure 5.4: Single-objective minima for different w

5.4 EPO

The Elliptical Pareto front Optimization routine (EPO) aims to use established and proven single objective methods to determine elliptical Pareto fronts. This a custom method developed here to try and exploit the elliptical nature exhibited by the multi-objective airfoil formulations. Linear aggregating functions [11] guide the optimization algorithm.

The method makes use of objective weights to guide a single objective optimizer to refine different sections of the Pareto front. The weights are generated by fitting a hyper ellipse to the designs in the repository. The EPO repository makes use of radial functions to determine crowding.

A weights vector w is used to cast the multi-objective function into a single objective function. Different w will steer the single objective optimizer to different sections of the Pareto front, as illustrated in Figure 5.4 . The single objective aggregating function f is

$$f(w, x) = w \cdot F(x). \quad (5.22)$$

When a new w is generated, it is done to target the oldest non-dominated design in the repository. The repository stores the non-dominated designs X_r , together with their corresponding objective-space values F_r , and ages A_r . After each single objective iteration i , A_r is increased. A new w is determined by finding the center of a hyper-ellipse fitted to F_r .

F_r is scaled into Y to increase the accuracy of the ellipse fit for poorly-scaled objective-function spaces:

$$Y = \{\mathbf{s} \otimes \mathbf{F} \text{ for } \mathbf{F} \in F_r\}, \quad (5.23)$$

where the scaling vector \mathbf{s} is calculated by determining the smallest hyper-rectangle that encompasses all F_r . The top corner \mathbf{t}^c and the bottom corner \mathbf{b}^c of the bounding hyper-rectangle are

$$t_j^c = \max\{F_j \text{ for } \mathbf{F} \in F_r\} \quad j = 1, 2, \dots, k \quad (5.24)$$

$$b_j^c = \min\{F_j \text{ for } \mathbf{F} \in F_r\} \quad j = 1, 2, \dots, k \quad (5.25)$$

giving

$$s_i = 1/(t_i^c - b_i^c) \quad j = 1, 2, \dots, k \quad (5.26)$$

Once the objective-space has been scaled, the hyper-ellipse can be fitted. An unrotated hyper-ellipse as a function of $y \in \Re^k$, has a radius component \mathbf{r}_i and a center component \mathbf{c}_i for each dimension. The hyper-ellipse formula is:

$$\sum_{i=1}^k \left(\frac{y_i - c_i}{r_i} \right)^2 = 1 \quad (5.27)$$

$$\therefore \sum_{i=1}^k \left(\frac{1}{r_i^2} y_i^2 - \frac{2c_i}{r_i^2} y_i + \frac{c_i^2}{r_i^2} \right) = 1. \quad (5.28)$$

Grouping the constant terms in (5.28) gives

$$\sum_{i=1}^k \left(\frac{1}{r_i^2} y_i^2 - \frac{2c_i}{r_i^2} y_i \right) + \Upsilon = 0. \quad (5.29)$$

Eliminating the scaling variants by dividing through by the constant Υ , generates the form that can be used for the least-square error fit:

$$\sum_{i=1}^k (a_i y_i^2 + b_i y_i) + 1 = 0. \quad (5.30)$$

The over determined linear system for fitting \mathbf{Y} , which has q observations, is given

by

$$\begin{bmatrix} Y_{1,1}^2 & Y_{2,1}^2 & \cdots & Y_{k,1}^2 & Y_{1,1} & Y_{2,1} & \cdots & Y_{k,1} \\ Y_{1,2}^2 & Y_{2,2}^2 & \cdots & Y_{k,2}^2 & Y_{1,2} & Y_{2,2} & \cdots & Y_{k,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ Y_{1,q}^2 & Y_{2,q}^2 & \cdots & Y_{k,q}^2 & Y_{1,q} & Y_{2,q} & \cdots & Y_{k,q} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \\ b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \\ -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} \quad (5.31)$$

$$\mathcal{A}\mathbf{E} = \mathcal{B}. \quad (5.32)$$

The least-square error solution to (5.32) is obtained from

$$\mathcal{A}^T \mathcal{A}\mathbf{E} = \mathcal{A}^T \mathcal{B}. \quad (5.33)$$

Once the linear system of order $2k$ is solved, c_i and r_i can be determined directly using the least squares solution. The quadratic rule applied to (5.30) on objective function dimension i gives

$$y_i = \frac{-b_i \pm \sqrt{b_i^2 - 4a_i \left[\sum_{j=1}^{k, j \neq i} (a_j y_j^2 + b_j y_j) + 1 \right]}}{2a_i}. \quad (5.34)$$

Inspecting (5.34) shows that c_i which is the value about which y_i is mirrored, is

$$c_i = \frac{-b_i}{2a_i}, \quad (5.35)$$

with the center of the ellipse \mathbf{e}_c fitted to F_r being:

$$e_{c,i} = c_i/s_i \quad i \in 1, 2, \dots, m \quad (5.36)$$

The center point formula (5.35) can be verified by substituting the value from (5.28) into it. Formally

$$\frac{-b_i}{2a_i} = \frac{-(-2c_i/r_i^2)/\Upsilon}{2(1/r_i^2)/\Upsilon} = c_i \quad (5.37)$$

The radius r_i , is determined by solving the maximization problem

$$r_i = \max \left(\frac{\sqrt{b_i^2 - 4a_i \left[\sum_{j=1}^{k, j \neq i} (a_j y_j^2 + b_j y_j) + 1 \right]}}{2a_i} \right). \quad (5.38)$$

For a hyper-ellipse we know the maximum of (5.38) will occur when all other $y_j = c_j$. Thus the radius component is

$$r_i = \frac{\sqrt{b_i^2 - 4a_i \left[\sum_{j=1}^{k, j \neq i} (a_j c_j^2 + b_j c_j) + 1 \right]}}{2a_i}. \quad (5.39)$$

The ellipse fitted to F_r is valid when

$$\text{all}(\mathbf{t}_c < \mathbf{e}_c). \quad (5.40)$$

\mathbf{w} is calculated to refine F_r around a target point \mathbf{f}^t as follows:

$$\mathbf{w} = \begin{cases} \text{normalized}(\mathbf{e}_c - \mathbf{f}^t) & \text{if (5.40) is valid,} \\ \text{normalized}(\mathbf{t}_c - \mathbf{f}^t) & \text{otherwise.} \end{cases} \quad (5.41)$$

\mathbf{f}^t is the oldest point in the repository. After a point is selected as the target point its age is reset. This mechanism aims to ensure that all sections on the Pareto front approximation receive attention.

If the Pareto front of the multi-objective problem does not resemble a hyper-ellipse which satisfies (5.40), then the EPO technique is not expected to work well. The pseudo-code for generating \mathbf{w} is given in Algorithm 12.

Algorithm 12 EPO pseudo code to generate \mathbf{w}

procedure EPO GENERATE $\mathbf{w}(F_r, A_r)$

\mathbf{t}_c ▷ (5.24)

\mathbf{b}_c ▷ (5.25)

\mathbf{f}^t ▷ oldest F_r according to A_r

if $\text{all}(\mathbf{t}_c - \mathbf{b}_c > 0)$ and can fit ellipse **then** ▷ eq. 5.33

\mathbf{e}_c ▷ (5.36)

\mathbf{w} ▷ (5.41)

else if any($\mathbf{t}_c - \mathbf{b}_c > 0$) **then**

$\mathbf{w} = \text{normalized}(\mathbf{t}_c - \mathbf{f}^t)$

else

 use old \mathbf{w} ▷ only one point on Pareto front

end if

end procedure

The size of the archive is limited to r_m by using a distance based crowding rule. The generalized crowding distance function \mathfrak{C} 's value for a point in the objective space, $\xi \in \mathfrak{R}^k$, is

$$\mathfrak{C}(\xi) = \sum \left\{ \frac{1}{\sum_{i=1}^k (s_k(\xi_i - F_i))^2} \forall F \in F_r \text{ if } \|\xi - F\| \neq 0 \right\}, \quad (5.42)$$

with the scaling factor \mathbf{s} the same as the one used for the hyper-ellipse fit (5.33).

The crowding function (5.42) is the summation of sub functions emitted from every design in the archive's objective value. Figure 5.5 shows a 2D example of the crowding function.

Since we need to determine which design in X_r has the highest crowding, only the crowding values of F_r are of interest. The repository's crowding value set C_r is

$$C_r = \{ \mathfrak{C}(F) \text{ for } F \in F_r \}. \quad (5.43)$$

Determining C_r directly as in (5.43) involves many repeated square distance calculations. This is due to symmetry in the distance calculations. The distance table \mathbf{C} is

$$\mathbf{C} = \begin{bmatrix} 0 & \text{sym.} & \dots \\ \delta_{1,2} & 0 & \dots \\ \delta_{1,3} & \delta_{2,3} & 0 & \dots \\ \delta_{1,4} & \delta_{2,4} & \delta_{3,4} & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \quad (5.44)$$

with

$$\delta_{i,j} = \begin{cases} \|F_{r,i} - F_{r,j}\|^{-2} & \text{if } \|F_{r,i} - F_{r,j}\| \neq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (5.45)$$

Note that C_r is determined by summing the columns of \mathbf{C} which requires $(\text{len}(F_r) - 1)/2 \times \text{len}(F_r)$ square-distance calculations. The pseudo code for adding a point to the EPO Repository is

Algorithm 13 EPO adding a point to the repository pseudo code

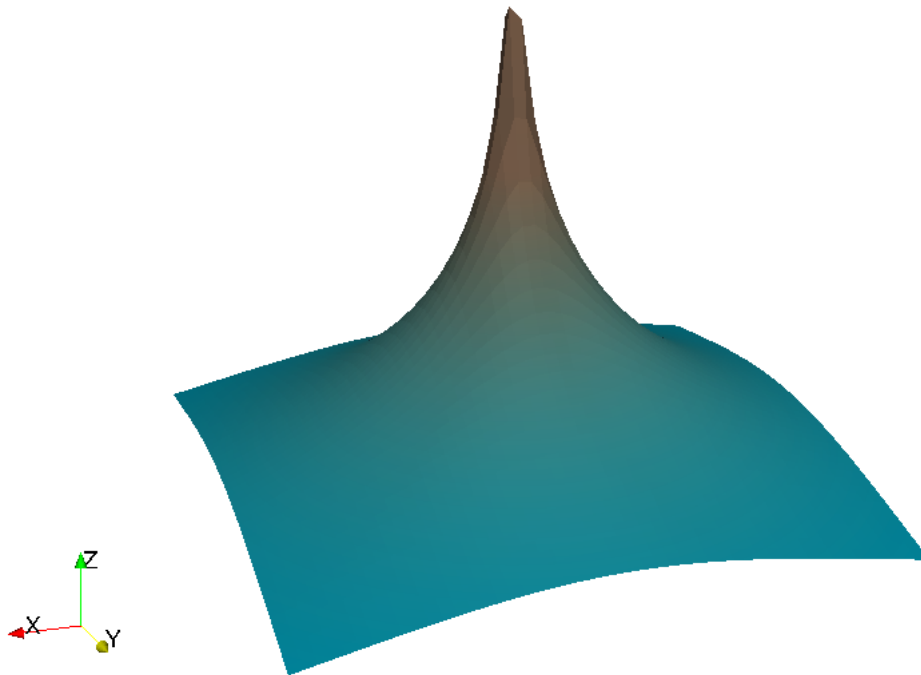
```

procedure REP INSPECT( $\mathbf{x}_c$ )
  if  $\mathbf{x}_c$  valid then                                     ▷ does not violate any constraints
    update  $X_r$                                            ▷ (5.6) and (5.7)
    if  $\text{len}(X_r) > r_m$  then
       $C$                                                  ▷ (5.43)
      remove the point with highest  $C$  from the repository.
    end if
  end if
end procedure

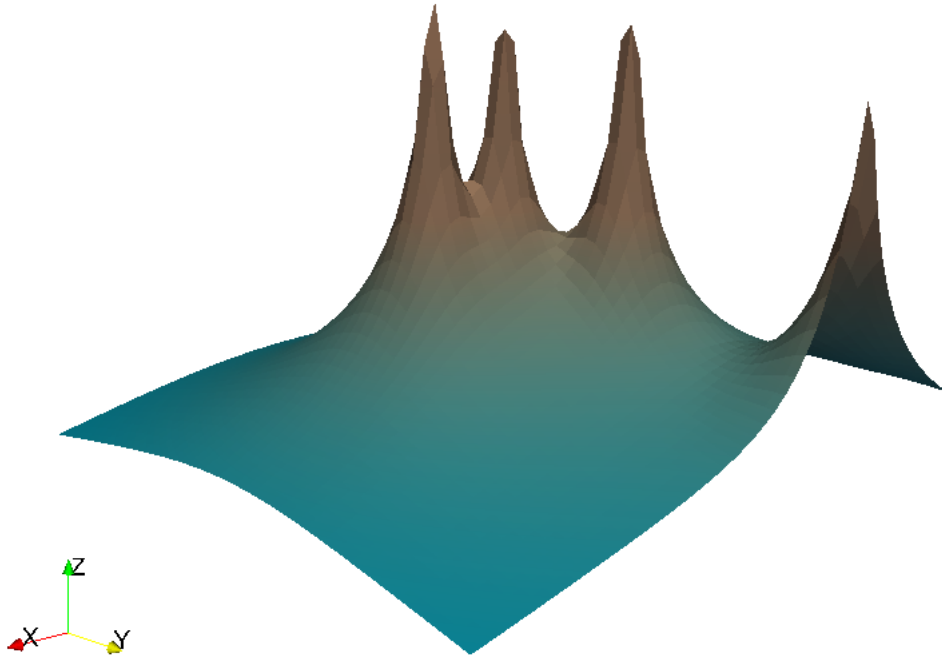
```

The EPO algorithm consists of main loop and a sub loop. In every iteration of the main loop:

- the objective blending weight vector (\mathbf{w}) is recalculated,



(a) Sub function emitted from designs in the archive



(b) Crowding function which is the summation of the sub functions

Figure 5.5: 2D illustration of the crowding function

- selected population member's designs are regenerated as to avoid diversity loss (shown later in (5.47)) and
- the single objective optimizer sub loop is run with the new \mathbf{w} .

The single objective optimizers used in this implementation are the PSO and DE population-based methods. Figure 5.6 show the expected EPO algorithm progression for a bi-objective function with an elliptical Pareto front.

The main loop continues a total of I_{max} times, resulting in I_{max} subproblems (single objective optimizations runs). I_{max} depends on the number of function evaluations allowed $evals$ and the number of iterations allocated to the single objective optimizer, i_{sub} . In the case of the PSO or DE single objective optimizers the maximum number of iterations is

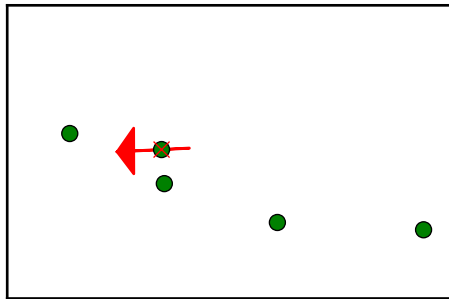
$$I_{max} = \left\lfloor \frac{evals}{N \cdot i_{sub}} \right\rfloor. \quad (5.46)$$

At the start of every main iteration calculated past the first, there is an α probability that a population member will be re-spawned around the target objective function value F^t . This is done to ensure that the diversity is maintained and also to speed up convergence to the aggregate function's suspected minimum \mathbf{x}^t . The target design \mathbf{f}^t is used as the center point for the new position. The size of the re-spawn hyper-rectangle is based upon the populating boundaries \mathbf{b}_l and \mathbf{b}_u and increases as I grows larger:

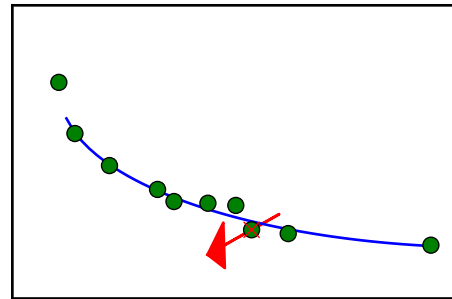
$$\mathbf{x}_0^j = \mathbf{x}^t + \frac{I}{I_{max}}(r_u(0, 1) - 0.5)(\mathbf{b}_u - \mathbf{b}_l). \quad (5.47)$$

If i_{sub} is very large, then the population of the single objective optimizer will all be close to the minimum of \mathbf{f} , and hence a large α is recommended. Alternatively, if i_{sub} is low then the population will still be widely distributed at the end of the sub-optimization, and a lower α should be used.

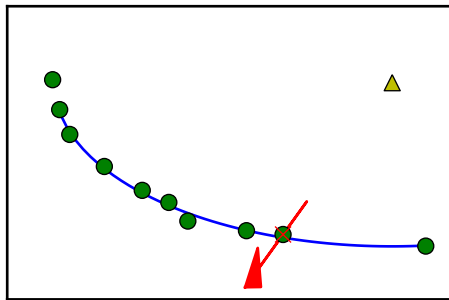
Every time the function is evaluated, the output is checked by the repository. The DE variant of EPO is given in Algorithm 14, and the PSO variant in Algorithm 15.



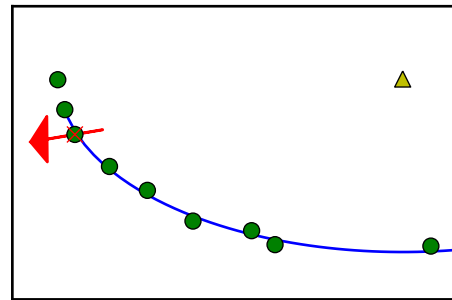
(a) Main iter. 2



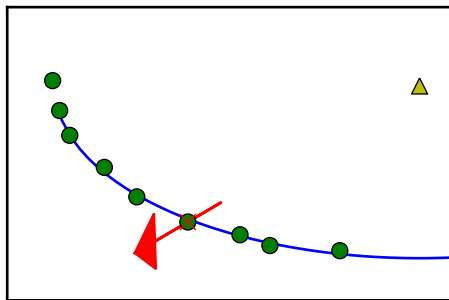
(b) Main iter. 3



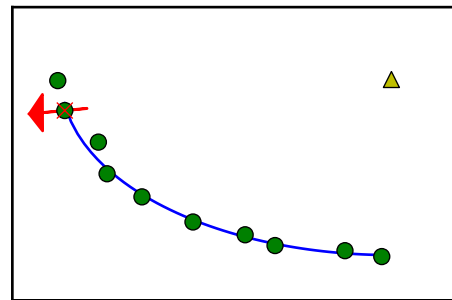
(c) Main iter. 6



(d) Main iter. 7



(e) Main iter. 8



(f) Main iter. 10

Figure 5.6: Progression of the EPO method under favorable circumstances. The cross shows f^t the design used to generate w whose ‘minimization’ direction is shown by the arrow. The line that represents the fitted ellipse and ellipse center (triangle) is only shown if the fit is valid.

Algorithm 14 EPODE pseudo code

```

procedure EPODE( $N, CR, F, I_{max}, i_{sub}, \alpha, \mathbf{b}_l, \mathbf{b}_u$ )
  for  $j \in \{0, 1, \dots, N - 1\}$  do                                     ▷ Initialize population
     $\mathbf{x}_0^j$                                                                                                      ▷ (4.1)
  end for
  for  $I \in \{0, 1, \dots, I_{max} - 1\}$  do
    calculate  $\mathbf{w}$                                                                                              ▷ Algorithm 12
    for  $i \in \{0, 1, \dots, i_{sub} - 1\}$  do
      for  $j \in \{0, 1, \dots, N - 1\}$  do
        if  $i = 0$  and  $I > 0$  and  $r_u(0, 1) < \alpha$  then
           $\mathbf{x}_0^j$                                                                                              ▷ (5.47)
        else
           $\mathbf{v}_{i,i}^j$                                                                                        ▷ (4.3)
           $\mathbf{u}_{i,i}^j$                                                                                        ▷ (4.5)
           $\mathbf{x}_i^j = \min(\mathbf{x}_{i-i}^j, \mathbf{u}_{i,i}^j)$                                                            ▷ Figure 4.1
        end if
      end for
      Increase age counter for each designs in repository.
    end for
  end for
end procedure

```

Algorithm 15 EPOPSO pseudo code

```

procedure EPOPSO( $N, \omega, c_1, c_2, I_{max}, i_{sub}, \alpha, \mathbf{b}_l, \mathbf{b}_u$ )
  for  $j \in \{0, 1, \dots, N - 1\}$  do                                     ▷ Initialize population
     $\mathbf{x}_0^j$                                                                                                      ▷ (4.1)
     $\mathbf{x}_{pb}^j = \mathbf{x}_0^j$ 
  end for
  for  $I \in \{0, 1, \dots, I_{max} - 1\}$  do
    calculate  $\mathbf{w}$                                                                                              ▷ Algorithm 12
    for  $i \in \{0, 1, \dots, i_{sub} - 1\}$  do
      for  $j \in \{0, 1, \dots, N - 1\}$  do
        if  $i = 0$  and  $I > 0$  and  $r_u(0, 1) < \alpha$  then
           $\mathbf{x}_0^j$                                                                                              ▷ (5.47)
           $\mathbf{x}_{pb}^j = \mathbf{x}_0^j$ 
           $\mathbf{v}_0^j = \mathbf{0}$ 
        else
           $\mathbf{v}_{i,i}^j$                                                                                        ▷ (4.8)
           $\mathbf{x}_{i,i}^j$                                                                                        ▷ (4.7)
           $\mathbf{x}_{pb}^j = \min(\mathbf{x}_{pb}^j, \mathbf{x}_{i,i}^j)$                                                            ▷ Figure 4.1
        end if
      end for
       $\mathbf{x}_{gb} = \min(\mathbf{x}_{pb}^1, \mathbf{x}_{pb}^2, \dots, \mathbf{x}_{pb}^N)$                                                            ▷ Figure 4.1
      Increase age counter for each designs in repository.
    end for
  end for
end procedure

```

5.5 Testing

Assessing the performance of multi-objective algorithms is a multi-objective problem in itself. This is as the criteria for assessing the non-dominated solutions are independent. For example, the closeness of a set of non-dominated designs to the true Pareto front is not coupled to the spread of this set. Zitzler [38] shows that the best way to determine the performance of a MOEA is to compare its results to another MOEA's results. In the tests performed here the MOEA solutions for the test functions are plotted for visual inspection as well as quantified using two performance indicators.

The quantitative performance indicators used are the generalized distance metric and the hyper volume metric [26]. The indicators compare the two Pareto front approximations. Since the Pareto fronts for the test functions are known, the comparison is done between the true Pareto front \mathcal{P} , and the Pareto front points approximated by the MOEA, P . The performance indicators are:

- The *generational distance* (GD) performance indicator returns the normalized sum of the points' in P shortest euclidean distance to \mathcal{P} ,

$$\text{GD} = \frac{\sqrt{\sum_{i=1}^n \mathbf{d}_i^2}}{n}, \quad (5.48)$$

where the distance \mathbf{d} is the distance vector which consists of n elements.

- The *Hyper-volume ratio* (HVR) performance indicator uses the ratio of the volume enclosed by the approximate Pareto front $V(P, \mathbf{p}_o)$, to the volume enclosed by the True Pareto front $V(\mathcal{P}, \mathbf{p}_o)$

$$\text{HVR} = 1 - \frac{V(P, \mathbf{p}_o)}{V(\mathcal{P}, \mathbf{p}_o)}, \quad (5.49)$$

where \mathbf{p}_o is the volume bounds. In this application \mathbf{p}_o is chosen such that

$$p_{o,i} = \max\{p_i \forall \mathbf{p} \in \mathcal{P}\}. \quad i \in 1, 2, \dots, k. \quad (5.50)$$

If an element in P lies beyond \mathbf{p}_o it is ignored. Figure 5.7 shows an example of the hyper volumes for a bi-objective function.

Four testing problems are used to assess the MOEA implementations. The first two are custom test problems which make use of the *constraint surface approach* [14], and the other test problems are from literature. The test problems are:

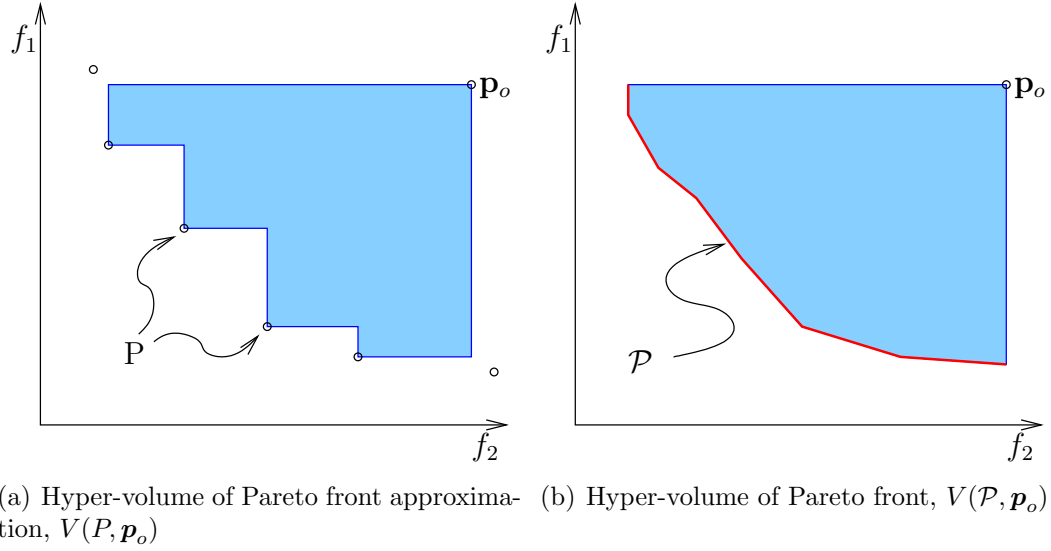


Figure 5.7: Hyper-volumes used by the HVR performance indicator for a bi-objective function.

- Multi-objective test problem 1: the line

$$F(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (5.51)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} 4 - x_1 - x_2 \\ -x_1 \\ -x_2 \end{bmatrix} \quad (5.52)$$

$$\mathbf{b}_l = [0.0, 0.0] \quad (5.53)$$

$$\mathbf{b}_u = [4.0, 4.0] \quad (5.54)$$

$$\mathcal{P} = \left\{ \begin{bmatrix} 4 - t \\ t \end{bmatrix} \forall t \in [0, 4] \right\} \quad (5.55)$$

The test function is basic with the objective space and design directly linked. It consists of only 2 design variables and 2 objective functions.

- Multi-objective test problem 2: the ellipse

$$F(x_1, x_2) = \begin{cases} [x_1, x_2]^T & \text{if } x < 2 \text{ or } x > 3 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (5.56)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} ((x_1 - 5)/4)^2 + ((x_2 - 1)/0.5)^2 - 1 \\ -x_1 \\ -x_2 \end{bmatrix} \quad (5.57)$$

$$\mathbf{b}_l = [0.0, 0.0] \quad (5.58)$$

$$\mathbf{b}_u = [4.0, 4.0] \quad (5.59)$$

$$\mathcal{P} = \left\{ \left[\begin{array}{c} t \\ -0.5\sqrt{1 - ((t - 5)/4)^2} + 1 \end{array} \right] \forall t \in [1, 2] + [3, 5] \right\} \quad (5.60)$$

The Pareto front for this test problem is two arcs of an ellipse. The undefined band adds the risk that the multi-objective optimizer could get stuck on one side of the undefined band and only solve part of the Pareto front.

- Multi-objective test problem 3: Kursawe as in [12]:

$$F(x_1, x_2, x_3) = \begin{bmatrix} \sum_{i=1}^{n-1} \left(-10e^{-0.2\sqrt{x_i^2 + x_{i+1}^2}} \right) \\ \sum_{i=1}^n (|x_i|^{0.8} + 5 \sin(x_i)^3) \end{bmatrix} \quad (5.61)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} -5 - x_1 \\ -5 - x_2 \\ -5 - x_3 \\ x_1 - 5 \\ x_2 - 5 \\ x_3 - 5 \end{bmatrix} \quad (5.62)$$

$$\mathbf{b}_l = [-5, -5, -5] \quad (5.63)$$

$$\mathbf{b}_u = [5, 5, 5] \quad (5.64)$$

$$\begin{aligned} \mathcal{P} \approx & \left\{ \begin{bmatrix} -20 \\ 0 \end{bmatrix} \right\} + \{F([t, 0, 0]^T) \forall t \in [-1.52, -0.51]\} \\ & + \{F([t, 0, t]^T) \forall t \in [-1.47, -1.0]\} \\ & + \{F([-1.52 + 0.25t, -1.52 + 0.72t, -1.52 + 0.25t]^T) \forall t \in [0, 1]\} \end{aligned} \quad (5.65)$$

- Multi-objective test problem 4: Deb as in [12]:

$$F(x_1, x_2, x_3) = \begin{bmatrix} x_1 \\ \frac{F_g(x_2)}{x_1} \end{bmatrix} \quad (5.66)$$

$$F_g(x_2) = 2.0 - e^{-((x_2-0.2)/0.004)^2} - 0.8e^{-((x_2-0.6)/0.4)^2} \quad (5.67)$$

$$g(x_1, x_2, x_3) = \begin{bmatrix} 0.1 - x_1 \\ 0.1 - x_2 \\ x_1 - 1 \\ x_2 - 1 \end{bmatrix} \quad (5.68)$$

$$\mathbf{b}_l = [0.1, 0.1] \quad (5.69)$$

$$\mathbf{b}_u = [1.0, 1.0] \quad (5.70)$$

$$\mathcal{P} = \left\{ \begin{bmatrix} t \\ F_g(0.2)/t \end{bmatrix} \forall t \in [0.1, 1] \right\} \quad (5.71)$$

When testing the multi-objective algorithms, the maximum allowable number of function evaluations was set to 15 000 and the repository size to 50. Each optimization was repeated 11 times to approximate MOEA consistency. Algorithm specific settings are summarized in Table 5.1.

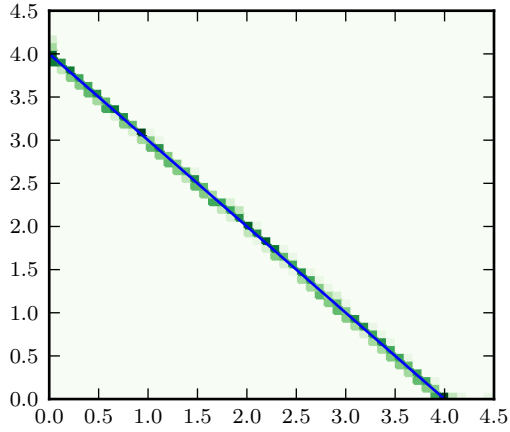
Note that the population members are not directly boxed inside the populating boundaries. This likely had a negative effect on the MOPSO algorithm which is designed with particle boxing built-in. Boxing is turned off to create similar conditions as those of the airfoil multi-objective formulations which do not have bound constraints.

The test results for the MOEAs on the testing problems are shown in Figures 5.8 to 5.11. Graphically it appears that all the MOEAs can successfully optimize the first three test problems. But the algorithms appear to struggle on the fourth test function with none consistently and accurately approximating the Pareto front.

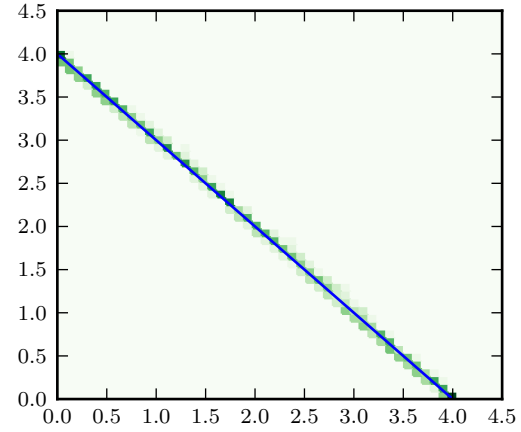
All algorithms managed to successfully approximate the Pareto front for the first test function. Table 5.2 shows the performance indicator value for the first test function, in which EPOPSO did the best according to the HVR measure. EPODE performed marginally worse than the EPOPSO algorithm, followed by the MOSADE algorithm.

| algorithm | settings |
|-----------|--|
| EPODE | $N = 50, F = 0.5, CR = 0.6, i_{sub} = 10, \alpha = 0.3$ |
| EPOPSO | $N = 50, \omega = 0.6, c_1 = 1, c_2 = 1, i_{sub} = 10, \alpha = 0.3$ |
| MOPSO | $N = 50, \omega = 0.4, \eta = 0.5$ |
| MOSADE | $N = 50, F_l = 0.1, F_u = 0.9, CR_l = 0.0, CR_u = 1.0$ |

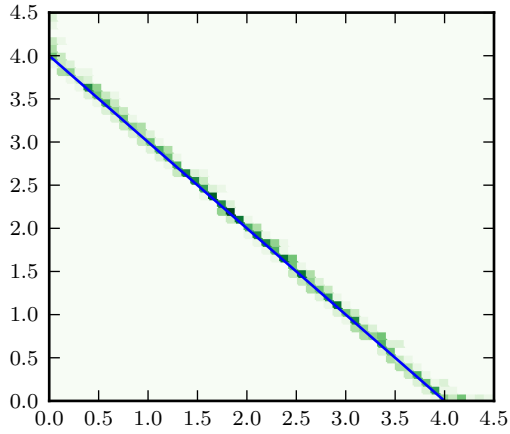
Table 5.1: MOEA settings for testing problems.



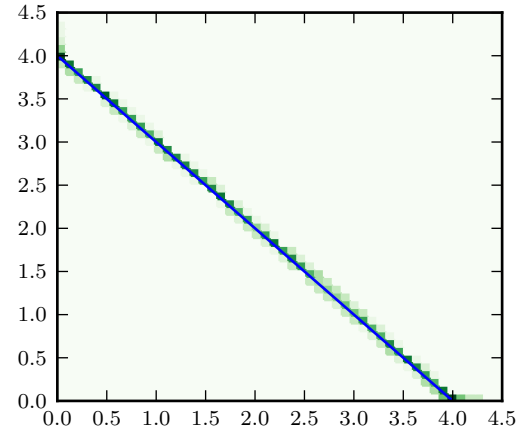
(a) EPODE



(b) EPOPSO



(c) MOPSO



(d) MOSADE

Figure 5.8: Probability density plot over 11 runs for the MOEA Pareto front approximations on the first test problem. The line in each diagram represents the true Pareto front.

| | \bar{n} | GD | | | HVR | | |
|--------|-----------|---------------|---------------|---------------|---------------|---------------|---------------|
| | | best | median | worst | best | median | worse |
| EPODE | 50.00 | 0.0021 | 0.0038 | 0.0896 | 0.0436 | 0.0626 | 0.0821 |
| EPOPSO | 50.00 | 0.0037 | 0.0046 | 0.0062 | 0.0383 | 0.0599 | 0.0728 |
| MOPSO | 50.00 | 0.0087 | 0.0486 | 0.1041 | 0.0556 | 0.0773 | 0.1318 |
| MOSADE | 50.00 | 0.0036 | 0.0058 | 0.0104 | 0.0443 | 0.0594 | 0.0782 |

Table 5.2: Performance indicators for MOEAs on the first test problem.

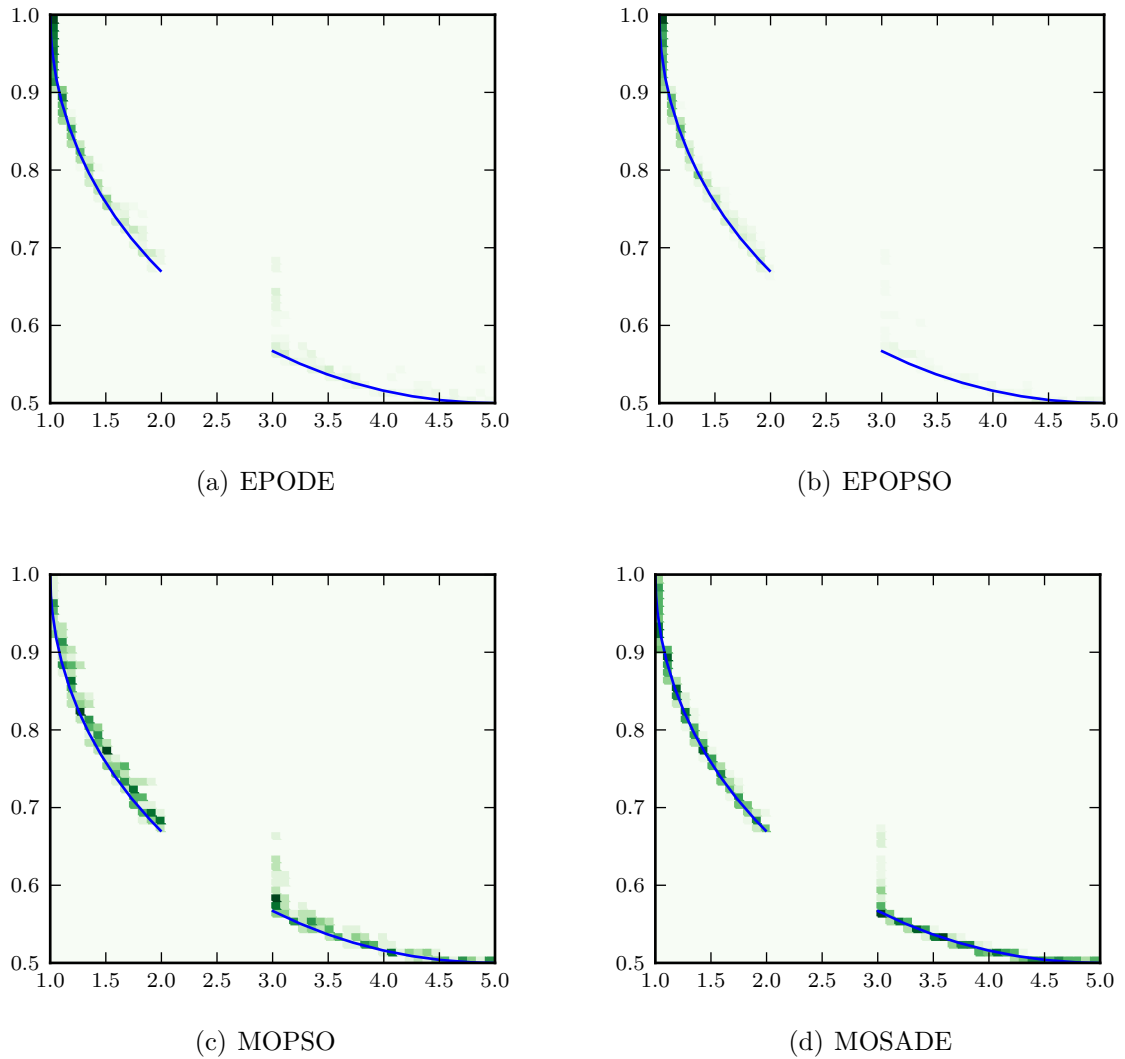


Figure 5.9: Probability density plot over 11 runs for the MOEA Pareto front approximations on the second test problem. The line in each diagram represents the true Pareto front.

| | \bar{n} | GD | | | HVR | | |
|--------|-----------|---------------|---------------|---------------|---------------|---------------|---------------|
| | | best | median | worst | best | median | worse |
| EPODE | 50.00 | 0.0012 | 0.0024 | 0.0030 | 0.0629 | 0.1760 | 0.3042 |
| EPOPSO | 50.00 | 0.0005 | 0.0013 | 0.0059 | 0.0779 | 0.3019 | 0.8730 |
| MOPSO | 31.09 | 0.0021 | 0.0046 | 0.0151 | 0.0857 | 0.1258 | 0.2261 |
| MOSADE | 50.00 | 0.0007 | 0.0017 | 0.0029 | 0.0545 | 0.0804 | 0.1103 |

Table 5.3: Performance indicators for MOEAs on the second test problem.

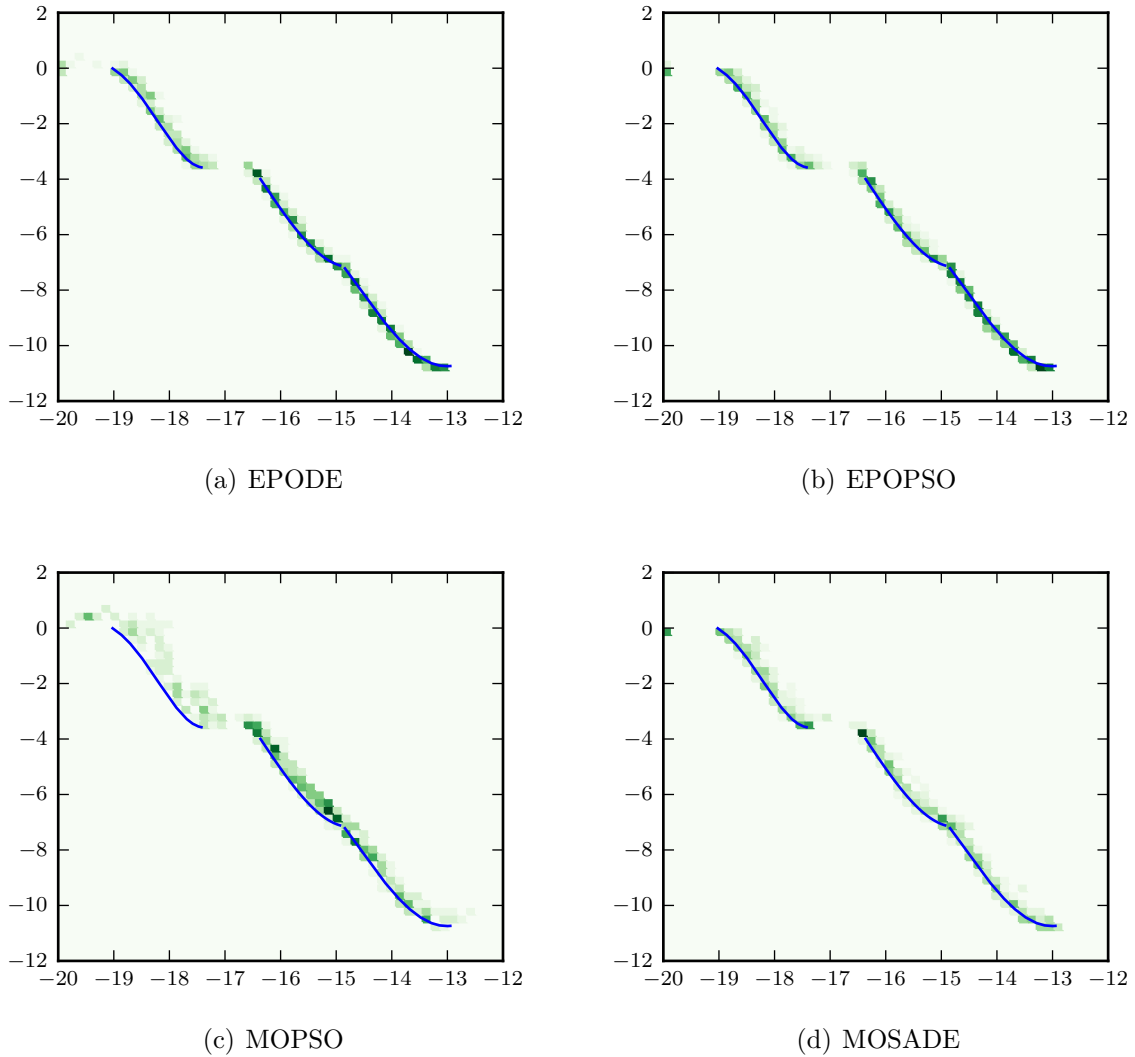


Figure 5.10: Probability density plot over 11 runs for the MOEA Pareto front approximations on the third test problem. The line in each diagram represents the true Pareto front.

| | \bar{n} | GD | | | HVR | | |
|--------|-----------|---------------|---------------|---------------|---------------|---------------|---------------|
| | | best | median | worst | best | median | worse |
| EPODE | 49.91 | 0.0106 | 0.0161 | 0.0230 | 0.0660 | 0.0864 | 0.1034 |
| EPOPSO | 49.82 | 0.0087 | 0.0143 | 0.0239 | 0.0663 | 0.0856 | 0.1004 |
| MOPSO | 36.91 | 0.0387 | 0.0506 | 0.0582 | 0.1217 | 0.2273 | 0.2902 |
| MOSADE | 44.45 | 0.0119 | 0.0209 | 0.0532 | 0.0565 | 0.1372 | 0.2876 |

Table 5.4: Performance indicators for MOEAs on the third test problem.

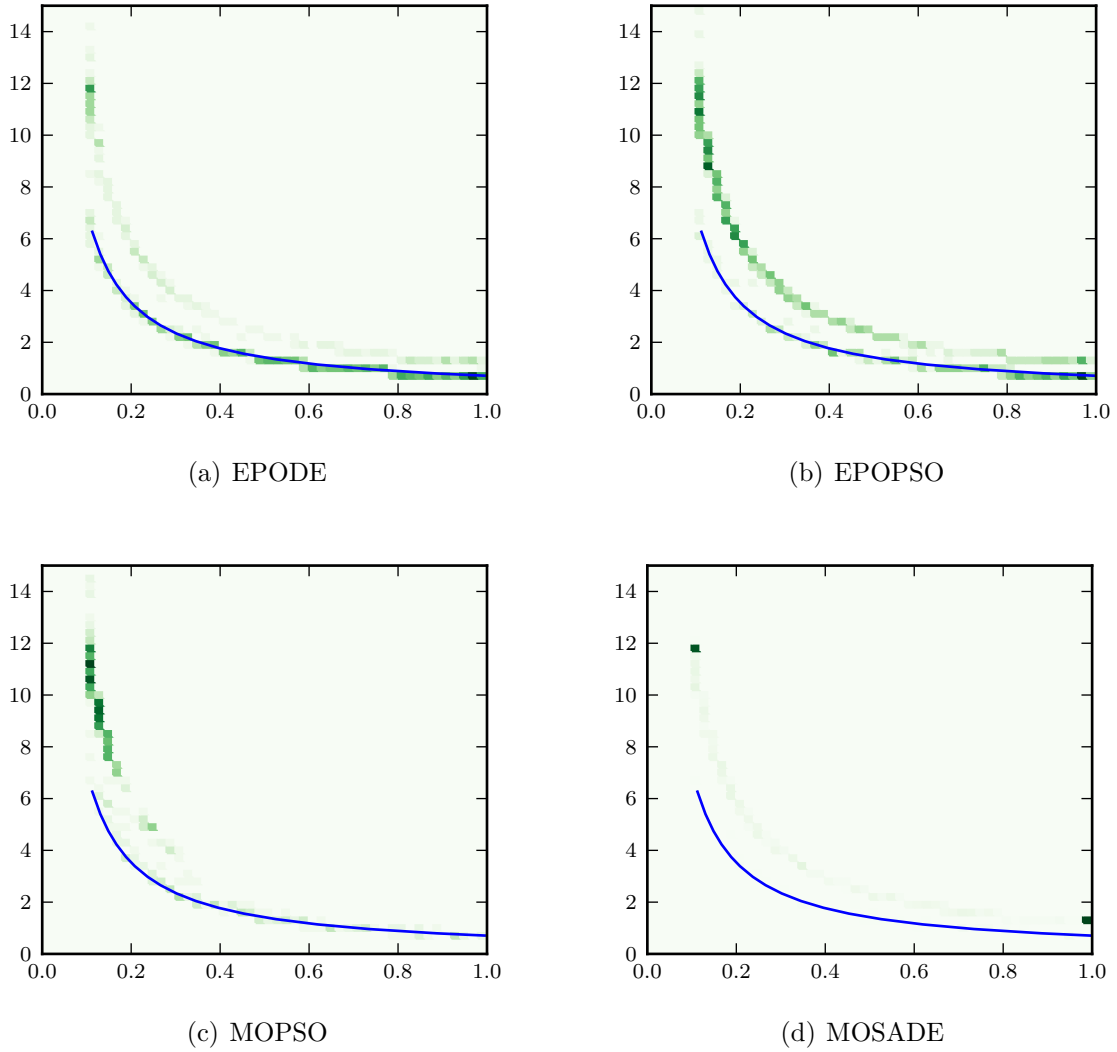


Figure 5.11: Probability density plot over 11 runs for the MOEA Pareto front approximations on the fourth test problem. The line in each diagram represents the true Pareto front.

| | \bar{n} | GD | | | HVR | | |
|--------|-----------|---------------|---------------|---------------|---------------|---------------|---------------|
| | | best | median | worst | best | median | worse |
| EPODE | 49.91 | 0.0964 | 0.1972 | 0.2847 | 0.0131 | 0.0281 | 0.2240 |
| EPOPSO | 49.91 | 0.2172 | 0.2603 | 0.3501 | 0.0115 | 0.1490 | 0.2235 |
| MOPSO | 39.73 | 0.2931 | 0.3856 | 0.6673 | 0.0642 | 0.1630 | 0.5289 |
| MOSADE | 48.73 | 0.2079 | 0.3006 | 0.4425 | 0.1554 | 0.2336 | 0.3439 |

Table 5.5: Performance indicators for MOEAs on the fourth fourth problem.

The MOPSO and EPODE algorithm have the largest variance on the GD metric. The MOPSO algorithm is the outlier for this test problem with its performance indicators significantly worse than the other MOEAs.

No algorithm clearly performed the best on the second test function according to the performance indicators shown in Table 5.3. What should be noted is that the MOPSO algorithm does not fill its repository to the maximum allocated size of 50, but averages a repository size of 31.

The performance indicator results for the third test problem are listed in Table 5.4. As for the second test problem, the EPO variants and the MOSADE algorithm have similar performance. The MOPSO implementation has the poorest performance indicators, and also does not fill the repository to its maximum size limit.

The performance indicators for the fourth function are shown in Table 5.5. The EPODE algorithm performs best on the GD indicator, and EPOPSO performs best on the HVR indicator.

The MOEAs have been implemented to a limited level of success. They are able to approximately determine the problem Pareto fronts but less successfully than the values reported for the original implementations. This poorer performance is probably caused by a minor implementation error or misinterpretation by the Author.

The airfoil multi-objective formulations are more difficult than the test functions used. The airfoil formulations each consists of 17 design variables and two or more objectives. But according to the MOEAs test results the implementations, although probably not operating at peak efficiency, should be able perform the airfoil multi-objective optimizations.

The airfoil multi-objective formulations are presented in the next section together with their optimization results.

5.6 Airfoil optimization

Three Pareto-optimal multi-objective formulations are chosen, each of which investigate different aspects of the physical problem. The first formulation is designed to investigate and quantify the effect of the avionics box height constraint. The second formulation has an objective for each of the three drag coefficients and the third formulation combines the first and second formulations. The airfoil Pareto-optimal multi-objective formulations are:

- Multi-objective formulation 1 (MOF1): avionics box height vs. blended drag coef-

ficients,

$$F(\mathbf{x}) = \begin{bmatrix} 3C_{D_1}(\mathbf{x}) + C_{D_2}(\mathbf{x}) + C_{D_3}(\mathbf{x}) \\ -B_h(\mathbf{x}) \end{bmatrix} \quad (5.72)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \max\text{Lift}(\mathbf{x}) \\ B_h(\mathbf{x}) - 100\text{mm} \end{bmatrix} \quad (5.73)$$

$$\mathbf{b}_l = [0, 0, \dots, 0] \quad (5.74)$$

$$\mathbf{b}_u = [1, 1, \dots, 1] \quad (5.75)$$

- Multi-objective formulation 2 (MOF2): cruise drag vs. loiter drag vs. high-speed dash drag for 50 % semi-span wing section,

$$F(\mathbf{x}) = \begin{bmatrix} C_{D_1}(\mathbf{x}) \\ C_{D_2}(\mathbf{x}) \\ C_{D_3}(\mathbf{x}) \end{bmatrix} \quad (5.76)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \max\text{Lift}(\mathbf{x}) \\ 73\text{mm}(\mathbf{x}) - B_h \end{bmatrix} \quad (5.77)$$

$$\mathbf{b}_l = [0, 0, \dots, 0] \quad (5.78)$$

$$\mathbf{b}_u = [1, 1, \dots, 1] \quad (5.79)$$

- Multi-objective formulation 3 (MOF3): avionics box height vs. cruise drag vs. loiter drag vs. high-speed dash drag,

$$F(\mathbf{x}) = \begin{bmatrix} C_{D_1}(\mathbf{x}) \\ C_{D_2}(\mathbf{x}) \\ C_{D_3}(\mathbf{x}) \\ -B_h(\mathbf{x}) \end{bmatrix} \quad (5.80)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \max\text{Lift}(\mathbf{x}) \\ B_h(\mathbf{x}) - 100\text{mm} \end{bmatrix} \quad (5.81)$$

$$\mathbf{b}_l = [0, 0, \dots, 0] \quad (5.82)$$

$$\mathbf{b}_u = [1, 1, \dots, 1] \quad (5.83)$$

The MOEA settings used are the same as those used in the testing section which are shown in Table 5.1, with the exception of the allowable function evaluations. The function evaluations required are expected to be larger than those of the *a priori* formulations, as a more detailed analysis is done. The number of function evaluations is set to 15 000 for MOF1, 20 000 for MOF2 and 40 000 for MOF3.

| avionics box height | MOF1 | <i>a priori</i> formulations | |
|---------------------|--------|------------------------------|----------------|
| | | population-based | gradient-based |
| 73 mm | 0.0365 | 0.0358 | 0.0367 |
| 10 mm | 0.0240 | 0.0192 | 0.0249 |

Table 5.6: The minimum blended drag values for two different avionics heights determined through optimization methods.

MOF1 investigates the effect that the avionics box’s height constraint has on the blended drag value. Determining this relationship assists the design engineer to quantify the cost associated with changing the avionics box height, as well as to give insight into how the shape of the Pareto optimal airfoil changes for different heights. MOF1 is also used for verification, since the results can be checked against the values obtained from the *a priori* formulations, whose solutions should lie on its Pareto front.

Figure 5.12 shows the results obtained from MOF1. The four designs plotted in this figure show the shape trend of the airfoil as the avionics boxes height is reduced. It appears that the shape trend mostly involves the thinning of the airfoil and changing the trailing edge angle.

The MOPSO algorithm clearly performed the best out of the MOEAs on MOF1. The majority of MOPSO’s approximated Pareto front dominates the approximations of the other algorithms. This is contrary to the performance on the test functions, where the MOPSO implementation was worse than the other MOEAs.

Checking the approximated Pareto front for MOF1 against the values obtained in the *a priori* optimization allows for the estimation of the accuracy of the approximated Pareto front. Table 5.6 shows the results from MOF1 together with the results obtained using the *a priori* formulations.

The approximated Pareto front dominates the values of the modified gradient-based and surrogate methods but not those from the population-based methods on the *a priori* formulations. The Pareto front distance error appears to be the largest for a thin avionics box decreasing as the avionics box becomes thicker. Increasing the maximum number of function evaluations to 20 000 or 30 000 should help to decrease these errors.

A multi-objective optimization presents the user with a family of solutions which they can choose from. Users can select the design/s which suit their criteria best. Appendix E shows the family of solutions determined for MOF1.

The MOPSO algorithm performed the best on MOF2 as well and generated the most non-dominated points when compared to the other MOEAs. To summarize,

- the MOPSO algorithm generated 53 designs,

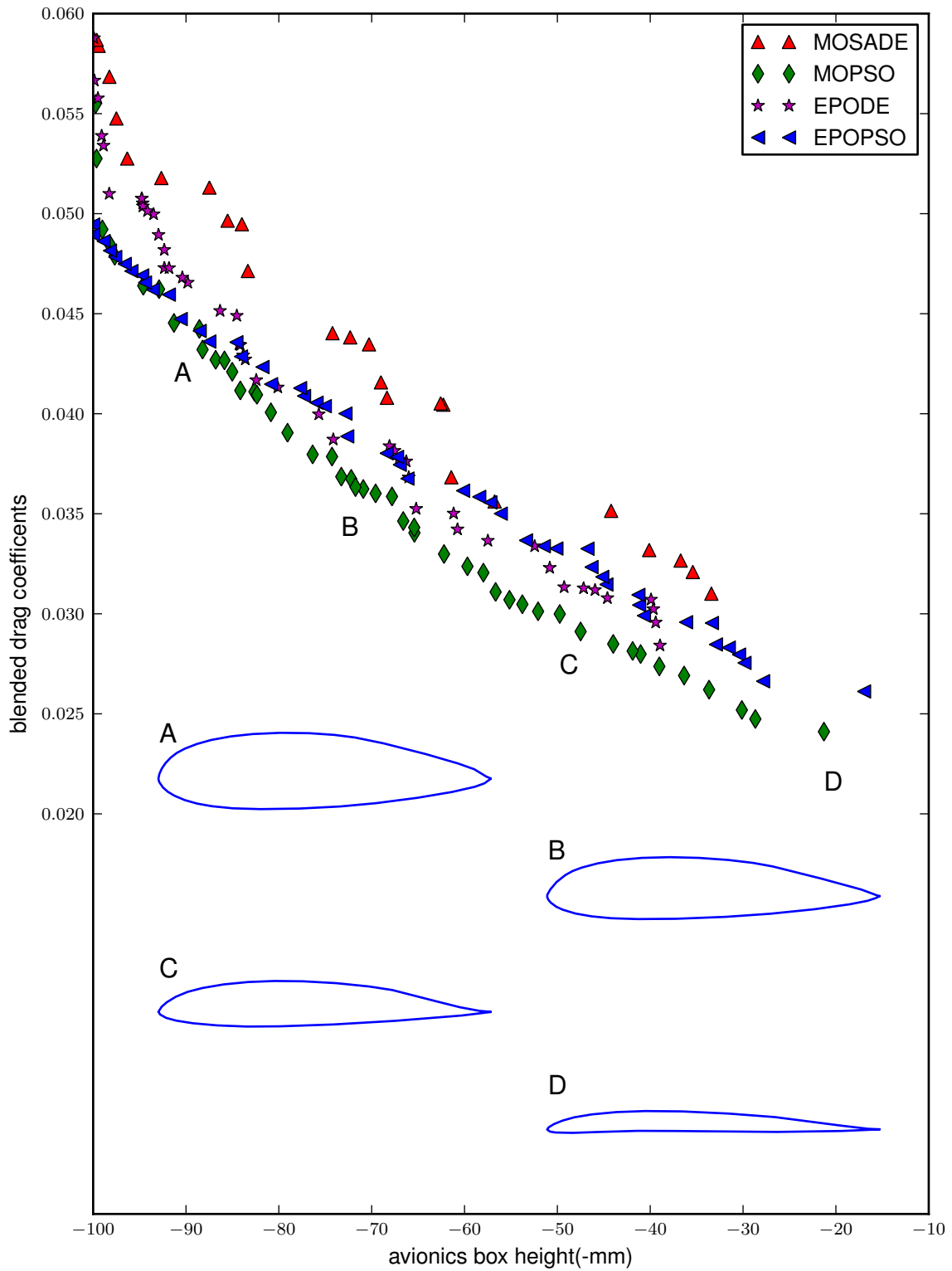
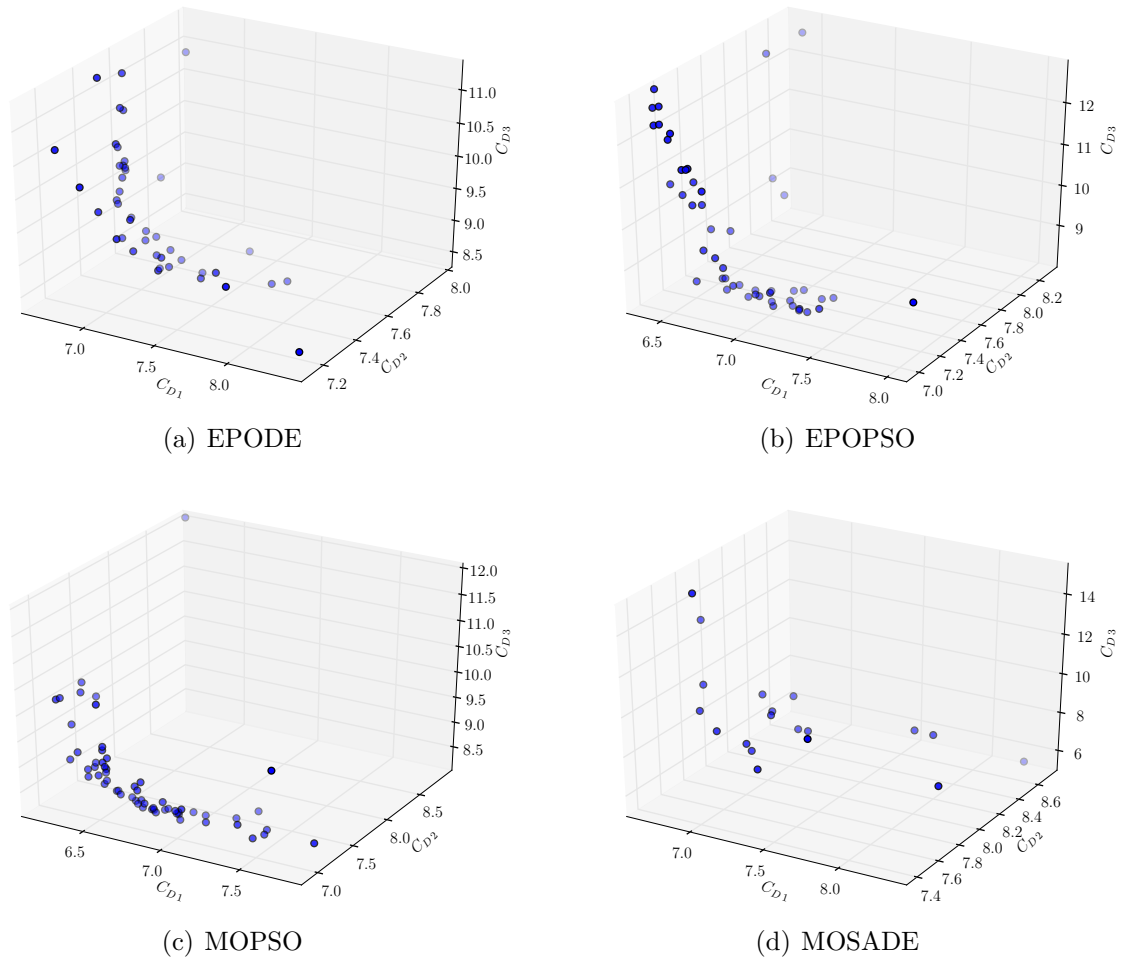


Figure 5.12: Results from avionics box height vs blended drag coefficients multi-objective formulation.



note: drag coefficient values are multiplied by 1000.

Figure 5.13: Multi-objective results for the minimization of three drag coefficients for different lift requirements.

- the EPOPSO algorithm generated 11 designs,
- the MOSADE algorithm generated one non dominated design and
- the EPODE algorithm generated no non dominated designs

As the number of objectives increase it becomes more difficult to visually gauge the MOEAs performance. The second formulation's results are shown in Figure 5.13.

At the start of the chapter the choice of blending weights used in the *a priori* formulations was questioned. The questions included what effect the blending weight had on the final design, and how would a design optimized for loiter or high-speed dash drag differ from one created using a blended drag. The results from MOF2 allow these questions to be answered.








| description | image | C_{D_1} (cruise) ($\times 10^2$) | C_{D_2} (loiter) ($\times 10^2$) | C_{D_3} (dash) ($\times 10^2$) |
|------------------------------|---|---|---|---------------------------------------|
| best dash |  | 0.896 | 1.200 | 0.613 |
| best cruise |  | 0.682 | 1.001 | 0.767 |
| best loiter |  | 0.877 | 0.515 | 0.824 |
| best for weights: {1,1,1} |  | 0.714 | 0.834 | 0.668 |
| best for weights: {2,1,1} |  | 0.691 | 0.896 | 0.648 |
| best for weights: {3,1,1} |  | 0.691 | 0.896 | 0.648 |
| best for weights: {4,1,1} |  | 0.691 | 0.896 | 0.648 |

Table 5.7: Selected designs from the approximated Pareto front for the second multi-objective formulation.

Designs selected from the solution to MOF2 are presented in Table 5.7. The Pareto front approximations give solutions for the best design if only one of the drag coefficients is considered. Comparison of the “best high-speed dash” and “best loiter” design show that both have a high cruise drag and that they differ largely in loiter and dash performance. The “best cruise” design appears to be more flattened out than the “best loiter” and “best high-speed dash” designs.

The airfoil seems to be relatively insensitive to the choice of blending weights used. The best solutions for the blending-weights of $\{2, 1, 1\}$, $\{3, 1, 1\}$ and $\{4, 1, 1\}$ are the same airfoil. The design with the minimum blended drag for the weights of $\{1, 1, 1\}$ does not differ significantly from the design corresponding to blending-weights of $\{3, 1, 1\}$.

The error in the Pareto front approximation is small. The design, from the family of solutions returned from MOF2, with the minimum blended drag for the weights $\{3, 1, 1\}$ has the blended drag value of 0.03617. This blended drag value is only 0.72% larger than the best solution obtained from the corresponding *a priori* formulations.

MOF3 results indicate that the MOEAs are unsuccessful in approximating the Pareto front. Visual inspection of the results, which are shown in Appendix G, shows this. Amongst the Pareto Front for MOF3 should be the results from MOF1, and MOF2. But these designs or equivalent designs are not present.

As the number of objective functions increase so does the Pareto Front complexity. MOF3 has an additional order of complexity due to the addition of the avionics box height objective function. It also appears that settings which are successful for MOF1 and MOF2 should be adjusted for MOF3. Perhaps allocating additional function evaluations, and increasing the repository size would improve the Pareto Front approximations.

The Pareto-optimal multiple objective optimizations provided the user with detailed insight which can be used to create a superior design. The Pareto-optimal multiple-objective methods are more costly than the *a priori* approach as they perform a more in depth analysis, but yield richer results.

CHAPTER 6

CONCLUSION

The optimization algorithms implemented have reproduced and extended the CSIR optimization results. The *a priori* avionics box height formulation results were reproduced and slightly improved, and the maximum lift formulations have been successfully implemented. Direct multi-objective optimization analysis has also been performed to supplement the findings.

The optimization problem's characteristics prevent gradient-based methods from successfully minimizing the *a priori* formulations. The various work-arounds of over-sampling, averaging gradients and excessive line searches were implemented to handle noise, poor gradient information and undefined space. Although the customized gradient methods managed to improve performance it came at the cost of many extra function evaluations, and it did not eliminate the algorithm parameter sensitivity. The gradient methods' inadequate performance is due to the optimization problem's characteristics that do not meet the prerequisites that gradient-based methods are developed to exploit.

The population-based methods are successful in optimizing the *a priori* formulations. For the avionics box height formulation they produce better designs in the high number of function evaluation domain. As for the maximum lift formulation, they produce significantly better designs as they are able to handle the bad characteristics inherent in the maximum lift constraint.

The Pareto-optimal multi-objective analysis provided further insight into the 2D UAV airfoil optimization problem. They have the advantage over the *a priori* approach as they allow the user to select a design(s) from the non-dominated designs set, removing the burden of having to assign importance to each objective before the optimization. The population-based multi-objective methods, just as the single objective methods, were first tested on sample problems to assure that they have been implemented successfully. Three different multi-objective formulations are implemented:

- the avionics box height vs. blended drag coefficient formulation,
- the cruise drag vs. the loiter drag vs. the high-speed dash drag formulation investigates the drag relationships and
- the third airfoil formulation combines the first and the second multi-objective formulations attempting to solve the drag performance for various avionics box heights.

The correct use of optimization techniques is important for a successful and competitive design process. For engineering application where trade-offs are common multi-objective optimization is especially important.

BIBLIOGRAPHY

- [1] Intel's Core 2 page. <http://www.intel.com/products/processor/core2duo/>.
- [2] Scientific Tools for Python, version 0.7.0. <http://www.scipy.org/>, 2009.
- [3] The Council for Scientific and Industrial Research (CSIR) in South Africa. <http://www.csir.co.za>, 2010.
- [4] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth Dover printing, tenth GPO printing edition, 1964.
- [5] A.J. Booker, J.E. Dennis, P.D. Frank, D.B. Serafini, V. Torczon, and M.W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural and Multidisciplinary Optimization*, 17(1):1–13, 1999.
- [6] B.A. Broughton and R. Heise. Optimisation of the Sekwa blended-wing-Body research UAV. In *Royal Aeronautical Society Annual Applied Aerodynamics Research Conference. London, UK*, pages 27–28, 2008.
- [7] B.A. Broughton, R.Heise, and D. Blaauw. Project Sekwa: A variable stability, blended-wing-body, research UAV. In *Royal Aeronautical Society Annual Applied Aerodynamics Research Conference. London, UK*, pages 27–28, 2008.
- [8] R. Burden and J. Faires. Numerical analysis, 8th Edition. *Thomson Higher Education, Belmont, CA, USA*, 2005.
- [9] K. Chiba, S. Obayashi, K. Nakahashi, and H. Morino. High-fidelity multidisciplinary design optimization of wing shape for regional jet aircraft. In *Evolutionary Multi-criterion Optimization*, pages 621–635. Springer, 2005.

- [10] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, Feb 2002.
- [11] C.A.C. Coello. Evolutionary multi-objective optimization: a historical view of the field. *Computational Intelligence Magazine, IEEE*, 1(1):28–36, Feb. 2006.
- [12] C.A.C. Coello, G.T. Pulido, and M.S. Lechuga. Handling multiple objectives with particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 8(3):256–279, June 2004.
- [13] J. Dahl and L. Vandenberghe. Cvxopt website. <http://abel.ee.ucla.edu/cvxopt/>, 2009.
- [14] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. *E-Commerce Technology, IEEE International Conference on*, 1: 825–830, 2002.
- [15] M. Drela. XFOIL- An analysis and design system for low Reynolds number airfoils. *Low Reynolds Number Aerodynamics*, pages 1–12, 1989.
- [16] M. Drela and H. Youngren. Xfoil 6.9 user primer. <http://raphael.mit.edu/xfoil/>, 2001.
- [17] R.A. Dunlap. *The golden ratio and Fibonacci numbers*. World Scientific, 1997.
- [18] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pages 39–43, Oct 1995.
- [19] W. Hock and K. Schittkowski. *Lecture Notes in Economics and Mathematical Systems 187*. Berlin-Heidelberg-New York: Springer Verlag, 1981.
- [20] A. Konak, D.W. Coit, and A.E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.
- [21] D. Kraft. Algorithm 733: TOMP–Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software (TOMS)*, 20(3):281, 1994.
- [22] C.L. Lawson and R.J. Hanson. *Solving least squares problems*. Society for Industrial Mathematics, 1995.
- [23] R.A.E. Makinen, J. Periaux, and J. Toivanen. Multidisciplinary shape optimization in aerodynamics and electromagnetics using genetic algorithms. *International Journal for Numerical Methods in Fluids*, 30(2):149–160, 1999.

- [24] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.
- [25] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 1999.
- [26] T. Okabe, Y. Jin, and B. Sendhoff. A critical survey of performance indices for multi-objective optimisation. In *Proc. of 2003 Congress on Evolutionary Computation*, pages 878–885. Citeseer, 2003.
- [27] M.J.D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis, Proceedings of the 6th Workshop on Optimization and Numerical Analysis*, volume 275, pages 51–67. Kluwer Academic Publishers, 1994.
- [28] M. Selig and M. Maughmer. Generalized multipoint inverse airfoil design. *AIAA journal*, 30(11):2618–2625, 1992.
- [29] M. Selig and M. Maughmer. Multipoint inverse airfoil design method based on conformal mapping. *AIAA journal*, 30(5):1162–1170, 1992. ISSN 0001-1452.
- [30] M.S. Selig. Overview of Profoil-www. <http://www.profoil.org/profoil/010-overview.html>, 2005.
- [31] J.A. Snyman. The lfopc leap-frog algorithm for constrained optimization. *Computers and Mathematics with Applications*, 40(8-9):1085 – 1096, 2000.
- [32] J.A. Snyman. *Practical Mathematical Optimization*. Springer, 2005.
- [33] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.
- [34] I.C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, 2003.
- [35] D.A.V. Veldhuizen and G.B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary computation*, 8(2):125–147, 2000.
- [36] Y.N. Wang, L.H. Wu, and X.F. Yuan. Multi-objective self-adaptive differential evolution with elitist archive and crowding entropy-based diversity measure. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, pages 193–209, January 2009.
- [37] D.N. Wilke, S. Kok, and A.A. Groenwold. Comparison of linear and classical velocity update rules in particle swarm optimization: notes on diversity. *International Journal for Numerical Methods in Engineering*, 70(8):962–984, 2007.

- [38] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, April 2003.

Appendix A - Further discussion on line search algorithm testing results

Figures 1 and 2 show additional detail regarding the line search behavior. The test functions used were:

$$t_1(x) = (x - 0.8)^2 \quad (1)$$

$$t_2(x) = t_1(x) - 0.2 \sin(\text{mod}(10(x - 0.8) + \pi/2), \pi) \quad (2)$$

$$t_3(x) = \begin{cases} t_1(x) & \text{if } \text{mod}(x, 0.6) > 0.3 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (3)$$

$$t_4(x) = \begin{cases} t_2(x) & \text{if } \text{mod}(x, 0.6) > 0.3 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (4)$$

For each function the starting point was $x = 0$, with a positive search direction and an initial section size of 1. The step tolerance was set to 10^{-6} . Figures 1 and 2 show the search performed by the golden section search and the populating section search respectively.

The performance of the golden section search on the second test function is explained as follows. In the first generation, the last point is the minimum and as such the search is expanded by adding a point at 1.618. In the next iteration the local minimum located at 1.1 causes the search section to be reduced incorrectly.

The golden section search reduces its search size quicker than the section populating search, but is less robust. The section populating search explores the space more thoroughly, which in simple cases such as the first test function is unnecessary. In more complicated scenarios such as the fourth test function and airfoil single objective optimization problems it is advantageous.

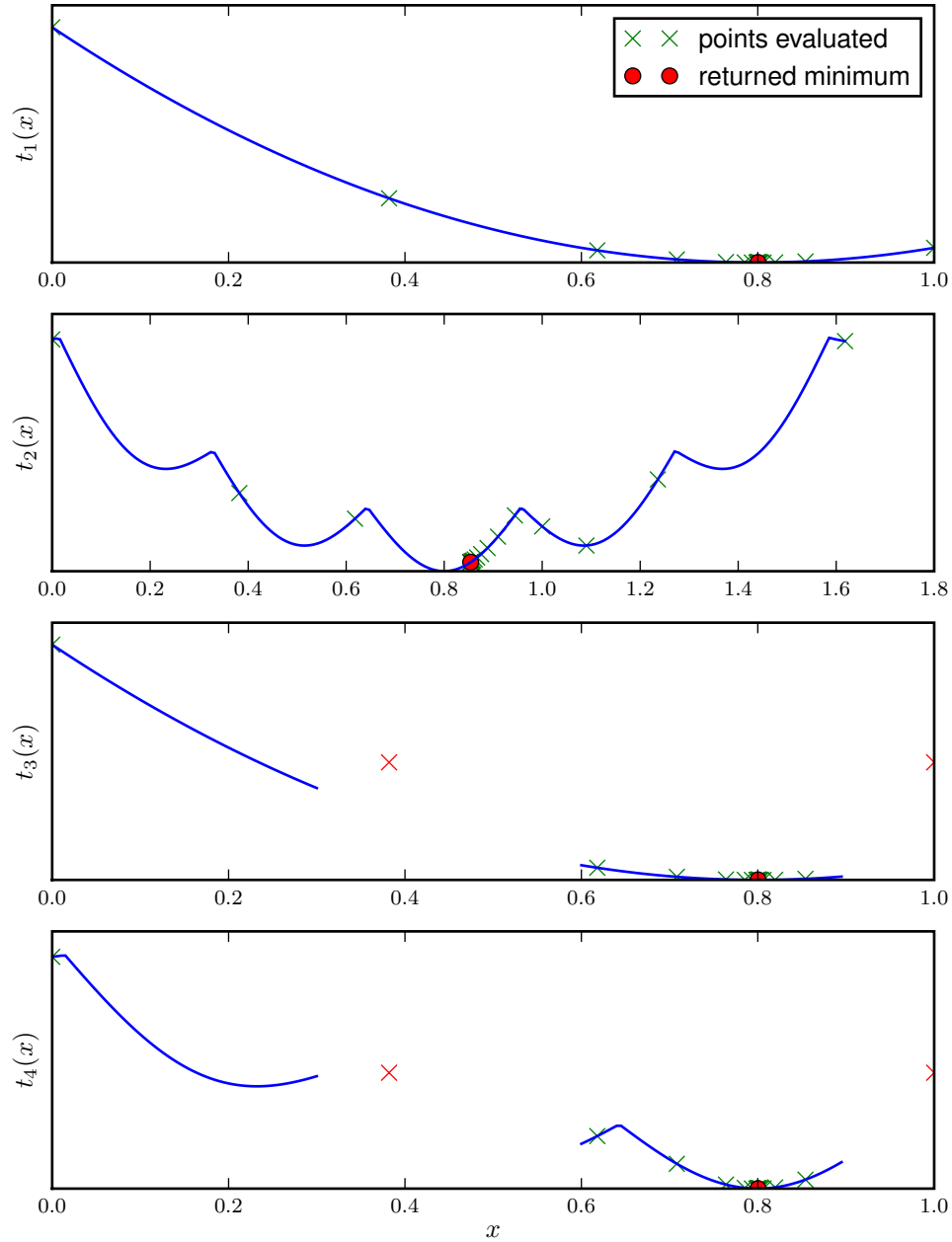


Figure 1: Golden section search, points evaluated during search

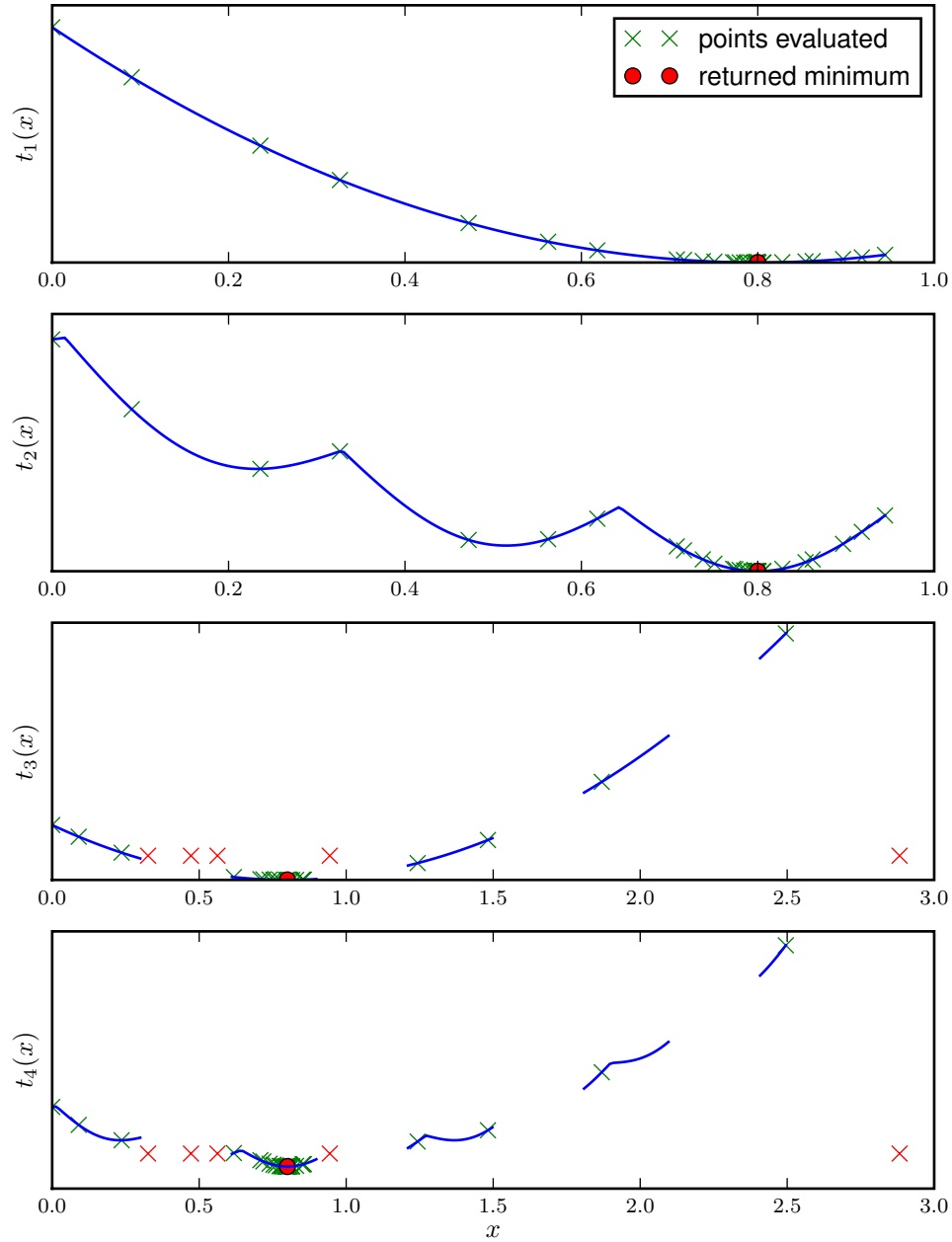


Figure 2: Section populating search, points evaluated during search

Appendix B - Constrained optimization single objective test functions

Selected problems from *Lecture Notes in Economics and Mathematical Systems 187* [19] where used to benchmark the gradient-based and population-based optimizers. The problem used are presented in this section.

- Test function 1 :

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (5)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} -1.5 - x_2 \end{bmatrix} \quad (6)$$

starting at \mathbf{x}^s [-2.0, 1.0] (feasible) , minimum at \mathbf{x}^* [1.0, 1.0]. $f(\mathbf{x}^s) = 909.0000$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 2 :

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (7)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} 1.5 - x_2 \end{bmatrix} \quad (8)$$

starting at \mathbf{x}^s [-2.0, 1.0] (infeasible) , minimum at \mathbf{x}^* [1.224371, 1.5]. $f(\mathbf{x}^s) = 909.0000$ and $f(\mathbf{x}^*) = 0.0504$

- Test function 3 :

$$f(x_1, x_2) = x_2 + 10.0^{-5}(x_2 - x_1)^2 \quad (9)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} -x_2 \end{bmatrix} \quad (10)$$

starting at \mathbf{x}^s [10.0, 1.0] (feasible) , minimum at \mathbf{x}^* [0.0, 0.0]. $f(\mathbf{x}^s) = 1.0008$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 4 :

$$f(x_1, x_2) = 1.0/3.0(x_1 + 1)^3 + x_2 \quad (11)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} 1 - x_1 \\ -x_2 \end{bmatrix} \quad (12)$$

starting at \mathbf{x}^s [1.125, 0.125] (feasible) , minimum at \mathbf{x}^* [1.0, 0.0]. $f(\mathbf{x}^s) = 3.3236$ and $f(\mathbf{x}^*) = 2.6667$

- Test function 5 :

$$f(x_1, x_2) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1 \quad (13)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} -1.5 - x_1 \\ x_1 - 4 \\ -3 - x_2 \\ x_2 - 3 \end{bmatrix} \quad (14)$$

starting at \mathbf{x}^s [0.0, 0.0] (feasible) , minimum at \mathbf{x}^* [-0.547198, -1.547198]. $f(\mathbf{x}^s) = 1.0000$ and $f(\mathbf{x}^*) = -1.9132$

- Test function 6 :

$$f(x_1, x_2) = (1 - x_1)^2 \quad (15)$$

$$\mathbf{h}(x_1, x_2) = \begin{bmatrix} 10(x_2 - x_1^2) \end{bmatrix} \quad (16)$$

starting at \mathbf{x}^s [-1.2, 1.0] (feasible) , minimum at \mathbf{x}^* [1.0, 1.0]. $f(\mathbf{x}^s) = 4.8400$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 7 :

$$f(x_1, x_2) = \log(1 + x_1^2) - x_2 \quad (17)$$

$$\mathbf{h}(x_1, x_2) = \begin{bmatrix} (1 + x_1^2)^2 + x_2^2 - 4 \end{bmatrix} \quad (18)$$

starting at \mathbf{x}^s [2.0, 2.0] (infeasible) , minimum at \mathbf{x}^* [0.0, 1.732051]. $f(\mathbf{x}^s) = -0.3906$ and $f(\mathbf{x}^*) = -1.7321$

- Test function 8 :

$$f(x_1, x_2) = -1.0 \quad (19)$$

$$\mathbf{h}(x_1, x_2) = \begin{bmatrix} x_1^2 + x_2^2 - 25 \\ x_1x_2 - 9 \end{bmatrix} \quad (20)$$

starting at \mathbf{x}^s [2.0, 1.0] (feasible) , minimum at \mathbf{x}^* [4.601595, 1.955844]. $f(\mathbf{x}^s) = -1.0000$ and $f(\mathbf{x}^*) = -1.0000$

- Test function 9 :

$$f(x_1, x_2) = \sin(\pi x_1/12) \cos(\pi x_2/16) \quad (21)$$

$$\mathbf{h}(x_1, x_2) = \begin{bmatrix} 4x_1 - 3x_2 \end{bmatrix} \quad (22)$$

starting at \mathbf{x}^s [0.0, 0.0] (feasible) , minimum at \mathbf{x}^* [-3.0, -4.0]. $f(\mathbf{x}^s) = 0.0000$ and $f(\mathbf{x}^*) = -0.5000$

- Test function 10 :

$$f(x_1, x_2) = x_1 - x_2 \quad (23)$$

$$\mathbf{g}(x_1, x_2) = \left[-(-3x_1^2 + 2x_1x_2 - x_2^2 + 1) \right] \quad (24)$$

starting at \mathbf{x}^s [-10.0, 10.0] (infeasible) , minimum at \mathbf{x}^* [0.0, 1.0]. $f(\mathbf{x}^s) = -20.0000$ and $f(\mathbf{x}^*) = -1.0000$

- Test function 11 :

$$f(x_1, x_2) = (x_1 - 5)^2 + x_2^2 - 25 \quad (25)$$

$$\mathbf{g}(x_1, x_2) = \left[-x_2 + x_1^2 \right] \quad (26)$$

starting at \mathbf{x}^s [4.9, 0.1] (infeasible) , minimum at \mathbf{x}^* [1.234741, 1.524584]. $f(\mathbf{x}^s) = -24.9800$ and $f(\mathbf{x}^*) = -8.4985$

- Test function 12 :

$$f(x_1, x_2) = 0.5x_1^2 + x_2^2 - x_1x_2 - 7x_1 - 7x_2 \quad (27)$$

$$\mathbf{g}(x_1, x_2) = \left[4x_1^2 + x_2^2 - 25 \right] \quad (28)$$

starting at \mathbf{x}^s [0.0, 0.0] (feasible) , minimum at \mathbf{x}^* [2.0, 3.0]. $f(\mathbf{x}^s) = 0.0000$ and $f(\mathbf{x}^*) = -30.0000$

- Test function 13 :

$$f(x_1, x_2) = (x_1 - 2)^2 + x_2^2 \quad (29)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} x_2 - (1 - x_1)^3 \\ -x_1 \\ -x_2 \end{bmatrix} \quad (30)$$

starting at \mathbf{x}^s [-2.0, -2.0] (infeasible) , minimum at \mathbf{x}^* [1.0, 0.0]. $f(\mathbf{x}^s) = 20.0000$ and $f(\mathbf{x}^*) = 1.0000$

- Test function 14 :

$$f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2 \quad (31)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} 0.25x_1^2 + x_2^2 - 1 \end{bmatrix} \quad (32)$$

$$\mathbf{h}(x_1, x_2) = \begin{bmatrix} -1 - x_1 + 2x_2 \end{bmatrix} \quad (33)$$

starting at \mathbf{x}^s [2.0, 2.0] (infeasible) , minimum at \mathbf{x}^* [0.822876, 0.911438]. $f(\mathbf{x}^s) = 1.0000$ and $f(\mathbf{x}^*) = 1.3935$

- Test function 15 :

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (34)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} 1 - x_1x_2 \\ -x_1 - x_2^2 \\ x_1 - 0.5 \end{bmatrix} \quad (35)$$

starting at \mathbf{x}^s [-2.0, 1.0] (infeasible) , minimum at \mathbf{x}^* [0.5, 2.0]. $f(\mathbf{x}^s) = 909.0000$ and $f(\mathbf{x}^*) = 306.5000$

- Test function 16 :

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (36)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} -x_1 - x_2^2 \\ -x_1^2 - x_2 \\ -0.5 - x_1 \\ x_1 - 0.5 \\ x_2 - 1 \end{bmatrix} \quad (37)$$

starting at \mathbf{x}^s [-2.0, 1.0] (infeasible) , minimum at \mathbf{x}^* [0.5, 0.25]. $f(\mathbf{x}^s) = 909.0000$ and $f(\mathbf{x}^*) = 0.2500$

- Test function 17 :

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (38)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} x_1 - x_2^2 \\ -x_1^2 + x_2 \\ -0.5 - x_1 \\ x_1 - 0.5 \\ x_2 - 1 \end{bmatrix} \quad (39)$$

starting at \mathbf{x}^s [-2.0, 1.0] (infeasible) , minimum at \mathbf{x}^* [0.0, 0.0]. $f(\mathbf{x}^s) = 909.0000$ and

$$f(\mathbf{x}^*) = 1.0000$$

- Test function 18 :

$$f(x_1, x_2) = 0.01x_1^2 + x_2^2 \quad (40)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} 25 - x_1x_2 \\ 25 - x_1^2 - x_2^2 \\ 2 - x_1 \\ x_1 - 50 \\ -x_2 \\ x_2 - 50 \end{bmatrix} \quad (41)$$

starting at \mathbf{x}^s [2.0, 2.0] (infeasible) , minimum at \mathbf{x}^* [15.811388, 1.581139]. $f(\mathbf{x}^s) = 4.0400$ and $f(\mathbf{x}^*) = 5.0000$

- Test function 19 :

$$f(x_1, x_2) = (x_1 - 10)^3 + (x_2 - 20)^3 \quad (42)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} 100 - (x_1 - 5)^2 - (x_2 - 5)^2 \\ -82.81 + (x_2 - 5)^2 + (x_1 - 6)^2 \\ 13 - x_1 \\ x_1 - 100 \\ -x_2 \\ x_2 - 100 \end{bmatrix} \quad (43)$$

starting at \mathbf{x}^s [20.1, 5.84] (infeasible) , minimum at \mathbf{x}^* [14.095000, 0.842961]. $f(\mathbf{x}^s) = -1808.8583$ and $f(\mathbf{x}^*) = -6961.8139$

- Test function 20 :

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (44)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} -x_1 - x_2^2 \\ -x_2 - x_1^2 \\ 1 - x_1^2 - x_2^2 \\ -0.5 - x_1 \\ x_1 - 0.5 \end{bmatrix} \quad (45)$$

starting at \mathbf{x}^s [-2.0, 1.0] (infeasible) , minimum at \mathbf{x}^* [0.5, 0.866025]. $f(\mathbf{x}^s) = 909.0000$ and $f(\mathbf{x}^*) = 38.1987$

- Test function 21 :

$$f(x_1, x_2) = 0.01x_1^2 + x_2^2 - 100 \quad (46)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} 10 - x_2 - 10x_1 \\ 2 - x_1 \\ x_1 - 50 \\ -50 - x_2 \\ x_2 - 50 \end{bmatrix} \quad (47)$$

starting at \mathbf{x}^s [-1.0, -1.0] (infeasible) , minimum at \mathbf{x}^* [2.0, 0.0]. $f(\mathbf{x}^s) = -98.9900$ and $f(\mathbf{x}^*) = -99.9600$

- Test function 22 :

$$f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2 \quad (48)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} x_1 + x_2 - 2 \\ x_1^2 - x_2 \end{bmatrix} \quad (49)$$

starting at \mathbf{x}^s [2.0, 2.0] (infeasible) , minimum at \mathbf{x}^* [1.0, 1.0]. $f(\mathbf{x}^s) = 1.0000$ and $f(\mathbf{x}^*) = 1.0000$

- Test function 23 :

$$f(x_1, x_2) = x_1^2 + x_2^2 \quad (50)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} -x_1 - x_2 + 1 \\ -x_1^2 - x_2^2 + 1 \\ -9x_1^2 - x_2^2 + 9 \\ -x_1^2 + x_2 \\ -x_2^2 + x_1 \\ -50 - x_1 \\ x_1 - 50 \\ -50 - x_2 \\ x_2 - 50 \end{bmatrix} \quad (51)$$

starting at \mathbf{x}^s [3.0, 1.0] (infeasible) , minimum at \mathbf{x}^* [1.0, 1.0]. $f(\mathbf{x}^s) = 10.0000$ and $f(\mathbf{x}^*) = 2.0000$

- Test function 24 :

$$f(x_1, x_2) = 1/(273^{0.5})((x_1 - 3)^2 - 9)x_2^3 \quad (52)$$

$$\mathbf{g}(x_1, x_2) = \begin{bmatrix} -x_1/3^{0.5} + x_2 \\ -x_1 - 3^{0.5}x_2 \\ x_1 + 3^{0.5}x_2 - 6 \\ -x_1 \\ -x_2 \end{bmatrix} \quad (53)$$

starting at \mathbf{x}^s [1.0, 0.5] (feasible) , minimum at \mathbf{x}^* [3.0, 1.732051]. $f(\mathbf{x}^s) = -0.0134$ and $f(\mathbf{x}^*) = -1.0000$

- Test function 25 :

$$f(x_1, x_2, x_3) = \sum_{i=1}^{100} f_i \quad (54)$$

$$\text{where } f_i = -0.01i + \exp(-1/x_1(u_i - x_2)^{x_3}) \quad (55)$$

$$\text{and } u_i = 25 + (-50 \log(0.01i))^{2/3} \quad (56)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} 0.1 - x_1 \\ x_1 - 100 \\ -x_2 \\ x_2 - 25.6 \\ -x_3 \\ x_3 - 5 \end{bmatrix} \quad (57)$$

starting at \mathbf{x}^s [100.0, 12.5, 3.0] (feasible) , minimum at \mathbf{x}^* [50.0, 25.0, 1.5]. $f(\mathbf{x}^s) = 32.8350$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 26 :

$$f(x_1, x_2, x_3) = (x_1 - x_2)^2 + (x_2 - x_3)^4 \quad (58)$$

$$\mathbf{h}(x_1, x_2, x_3) = \left[(1 + x_2^2)x_1 + x_3^4 - 3 \right] \quad (59)$$

starting at \mathbf{x}^s [-2.6, 2.0, 2.0] (feasible) , minimum at \mathbf{x}^* (multiple) [-1.810536, -1.810536, -1.810536],[1.0, 1.0, 1.0]. $f(\mathbf{x}^s) = 21.1600$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 27 :

$$f(x_1, x_2, x_3) = 0.01(x_1 - 1)^2 + (x_2 - x_1^2)^2 \quad (60)$$

$$\mathbf{h}(x_1, x_2, x_3) = \left[x_1 + x_3^2 + 1 \right] \quad (61)$$

starting at \mathbf{x}^s [2.0, 2.0, 2.0] (infeasible) , minimum at \mathbf{x}^* [-1.0, 1.0, 0.0]. $f(\mathbf{x}^s) = 4.0100$
and $f(\mathbf{x}^*) = 0.0400$

- Test function 28 :

$$f(x_1, x_2, x_3) = (x_1 + x_2)^2 + (x_2 + x_3)^2 \quad (62)$$

$$\mathbf{h}(x_1, x_2, x_3) = \begin{bmatrix} x_1 + 2x_2 + 3x_3 - 1 \end{bmatrix} \quad (63)$$

starting at \mathbf{x}^s [-4.0, 1.0, 1.0] (feasible) , minimum at \mathbf{x}^* [0.5, -0.5, 0.5]. $f(\mathbf{x}^s) = 13.0000$
and $f(\mathbf{x}^*) = 0.0000$

- Test function 29 :

$$f(x_1, x_2, x_3) = -x_1x_2x_3 \quad (64)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} x_1^2 + 2x_2^2 + 4x_3^2 - 48 \end{bmatrix} \quad (65)$$

starting at \mathbf{x}^s [1.0, 1.0, 1.0] (feasible) , minimum at \mathbf{x}^* [4.0, 2.828427, 2.0]. $f(\mathbf{x}^s) = -1.0000$
and $f(\mathbf{x}^*) = -22.6274$

- Test function 30 :

$$f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 \quad (66)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} -x_1^2 - x_2^2 + 1 \\ 1 - x_1 \\ x_1 - 10 \\ -10 - x_2 \\ x_2 - 10 \\ -10 - x_3 \\ x_3 - 10 \end{bmatrix} \quad (67)$$

starting at \mathbf{x}^s [1.0, 1.0, 1.0] (feasible) , minimum at \mathbf{x}^* [1.0, 0.0, 0.0]. $f(\mathbf{x}^s) = 3.0000$
and $f(\mathbf{x}^*) = 1.0000$

- Test function 31 :

$$f(x_1, x_2, x_3) = 9x_1^2 + x_2^2 + 9x_3^2 \quad (68)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} 1 - x_1x_2 \\ -10 - x_1 \\ x_1 - 10 \\ 1 - x_2 \\ x_2 - 10 \\ -10 - x_3 \\ x_3 - 1 \end{bmatrix} \quad (69)$$

starting at \mathbf{x}^s [1.0, 1.0, 1.0] (feasible) , minimum at \mathbf{x}^* [0.577350, 1.732051, 0.0].
 $f(\mathbf{x}^s) = 19.0000$ and $f(\mathbf{x}^*) = 6.0000$

- Test function 32 :

$$f(x_1, x_2, x_3) = (x_1 + 3x_2 + x_3)^2 + 4(x_1 - x_2)^2 \quad (70)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} -6x_2 - 4x_3 + x_1^3 + 3 \\ -x_1 \\ -x_2 \\ -x_3 \end{bmatrix} \quad (71)$$

$$\mathbf{h}(x_1, x_2, x_3) = [1 - x_1 - x_2 - x_3] \quad (72)$$

starting at \mathbf{x}^s [0.1, 0.7, 0.2] (feasible) , minimum at \mathbf{x}^* [0.0, 0.0, 1.0]. $f(\mathbf{x}^s) = 7.2000$
and $f(\mathbf{x}^*) = 1.0000$

- Test function 33 :

$$f(x_1, x_2, x_3) = (x_1 - 1)(x_1 - 2)(x_1 - 3) + x_3 \quad (73)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} -x_3^2 + x_2^2 + x_1^2 \\ -x_1^2 - x_2^2 - x_3^2 + 4 \\ -x_1 \\ -x_2 \\ -x_3 \\ x_3 - 5 \end{bmatrix} \quad (74)$$

starting at \mathbf{x}^s [0.0, 0.0, 3.0] (feasible) , minimum at \mathbf{x}^* [0.0, 1.414214, 1.414214].
 $f(\mathbf{x}^s) = -3.0000$ and $f(\mathbf{x}^*) = -4.5858$

- Test function 34 :

$$f(x_1, x_2, x_3) = -x_1 \quad (75)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} -x_2 + \exp(x_1) \\ -x_3 + \exp(x_2) \\ -x_1 \\ x_1 - 100 \\ -x_2 \\ x_2 - 100 \\ -x_3 \\ x_3 - 10 \end{bmatrix} \quad (76)$$

starting at \mathbf{x}^s [0.0, 1.05, 2.9] (feasible) , minimum at \mathbf{x}^* [0.834032, 2.302585, 10.0].
 $f(\mathbf{x}^s) = -0.0000$ and $f(\mathbf{x}^*) = -0.8340$

- Test function 35 :

$$f(x_1, x_2, x_3) = 9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2 + 2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3 \quad (77)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} -3 + x_1 + x_2 + 2x_3 \\ -x_1 \\ -x_2 \\ -x_3 \end{bmatrix} \quad (78)$$

starting at \mathbf{x}^s [0.5, 0.5, 0.5] (feasible) , minimum at \mathbf{x}^* [1.333333, 0.777778, 0.444444].
 $f(\mathbf{x}^s) = 2.2500$ and $f(\mathbf{x}^*) = 0.1111$

- Test function 36 :

$$f(x_1, x_2, x_3) = -x_1x_2x_3 \quad (79)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} -72 + x_1 + 2x_2 + 2x_3 \\ -x_1 \\ x_1 - 20 \\ -x_2 \\ x_2 - 11 \\ -x_3 \\ x_3 - 42 \end{bmatrix} \quad (80)$$

starting at \mathbf{x}^s [10.0, 10.0, 10.0] (feasible) , minimum at \mathbf{x}^* [20.0, 11.0, 15.0]. $f(\mathbf{x}^s) = -1000.0000$ and $f(\mathbf{x}^*) = -3300.0000$

- Test function 37 :

$$f(x_1, x_2, x_3) = -x_1x_2x_3 \quad (81)$$

$$\mathbf{g}(x_1, x_2, x_3) = \begin{bmatrix} -72 + x_1 + 2x_2 + 2x_3 \\ -x_1 - 2x_2 - 2x_3 \\ -x_1 \\ -x_2 \\ -x_3 \\ x_1 - 42 \\ x_2 - 42 \\ x_3 - 42 \end{bmatrix} \quad (82)$$

starting at \mathbf{x}^s [10.0, 10.0, 10.0] (feasible) , minimum at \mathbf{x}^* [24.0, 12.0, 12.0]. $f(\mathbf{x}^s) = -1000.0000$ and $f(\mathbf{x}^*) = -3456.0000$

- Test function 38 :

$$f(x_1, x_2, x_3, x_4) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \quad (83)$$

$$\mathbf{g}(x_1, x_2, x_3, x_4) = \begin{bmatrix} -10 - x_1 \\ -10 - x_2 \\ -10 - x_3 \\ -10 - x_4 \\ x_1 - 10 \\ x_2 - 10 \\ x_3 - 10 \\ x_4 - 10 \end{bmatrix} \quad (84)$$

starting at \mathbf{x}^s [-3.0, -1.0, -3.0, -1.0] (feasible) , minimum at \mathbf{x}^* [1.0, 1.0, 1.0, 1.0]. $f(\mathbf{x}^s) = 19192.0000$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 39 :

$$f(x_1, x_2, x_3, x_4) = -x_1 \quad (85)$$

$$\mathbf{g}(x_1, x_2, x_3, x_4) = \begin{bmatrix} -x_2 + x_1^3 + x_3^2 \\ -x_1^2 + x_2 + x_4^2 \end{bmatrix} \quad (86)$$

starting at \mathbf{x}^s [2.0, 2.0, 2.0, 2.0] (infeasible) , minimum at \mathbf{x}^* [1.0, 1.0, 0.0, 0.0]. $f(\mathbf{x}^s) = -2.0000$ and $f(\mathbf{x}^*) = -1.0000$

- Test function 40 :

$$f(x_1, x_2, x_3, x_4) = -x_1x_2x_3x_4 \quad (87)$$

$$\mathbf{h}(x_1, x_2, x_3, x_4) = \begin{bmatrix} -x_1^3 - x_2^2 + 1 \\ -x_1^2x_4 + x_3 \\ -x_4^2 + x_2 \end{bmatrix} \quad (88)$$

starting at \mathbf{x}^s [0.8, 0.8, 0.8, 0.8] (infeasible) , minimum at \mathbf{x}^* [0.793701, 0.707107, 0.529732, 0.840896]. $f(\mathbf{x}^s) = -0.4096$ and $f(\mathbf{x}^*) = -0.2500$

- Test function 41 :

$$f(x_1, x_2, x_3, x_4) = 2 - x_1x_2x_3 \quad (89)$$

$$\mathbf{g}(x_1, x_2, x_3, x_4) = \begin{bmatrix} -x_1 \\ -x_2 \\ -x_3 \\ x_1 - 1 \\ x_2 - 1 \\ x_3 - 1 \\ -x_4 \\ x_4 - 2 \end{bmatrix} \quad (90)$$

$$\mathbf{h}(x_1, x_2, x_3, x_4) = [x_1 + 2x_2 + 2x_3 - x_4] \quad (91)$$

starting at \mathbf{x}^s [2.0, 2.0, 2.0, 2.0] (infeasible) , minimum at \mathbf{x}^* [0.666667, 0.333333, 0.333333, 2.0]. $f(\mathbf{x}^s) = -6.0000$ and $f(\mathbf{x}^*) = 1.9259$

- Test function 42 :

$$f(x_1, x_2, x_3, x_4) = (x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 - 3)^2 + (x_4 - 4)^2 \quad (92)$$

$$\mathbf{h}(x_1, x_2, x_3, x_4) = \begin{bmatrix} x_1 - 2 \\ x_3^2 + x_4^2 - 2 \end{bmatrix} \quad (93)$$

starting at \mathbf{x}^s [1.0, 1.0, 1.0, 1.0] (feasible) , minimum at \mathbf{x}^* [2.0, 2.0, 0.848528, 1.131371]. $f(\mathbf{x}^s) = 14.0000$ and $f(\mathbf{x}^*) = 13.8579$

- Test function 43 :

$$f(x_1, x_2, x_3, x_4) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4 \quad (94)$$

$$\mathbf{g}(x_1, x_2, x_3, x_4) = \begin{bmatrix} -8 + x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 \\ -10 + x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 \\ -5 + 2x_1^2 + x_2^2 + x_3^2 + 2x_1 - x_2 - x_4 \end{bmatrix} \quad (95)$$

starting at \mathbf{x}^s [0.0, 0.0, 0.0, 0.0] (feasible) , minimum at \mathbf{x}^* [0.0, 1.0, 2.0, -1.0]. $f(\mathbf{x}^s) = 0.0000$ and $f(\mathbf{x}^*) = -44.0000$

- Test function 44 :

$$f(x_1, x_2, x_3, x_4) = x_1 - x_2 - x_3 - x_1x_3 + x_1x_4 + x_2x_3 - x_2x_4 \quad (96)$$

$$\mathbf{g}(x_1, x_2, x_3, x_4) = \begin{bmatrix} -8 + x_1 + 2x_2 \\ -12 + 4x_1 + x_2 \\ -12 + 3x_1 + 4x_2 \\ -8 + 2x_3 + x_4 \\ -8 + x_3 + 2x_4 \\ -5 + x_3 + x_4 \\ -x_1 \\ -x_2 \\ -x_3 \\ -x_4 \end{bmatrix} \quad (97)$$

starting at \mathbf{x}^s [0.0, 0.0, 0.0, 0.0] (feasible) , minimum at \mathbf{x}^* [0.0, 3.0, 0.0, 4.0]. $f(\mathbf{x}^s) = 0.0000$ and $f(\mathbf{x}^*) = -15.0000$

- Test function 45 :

$$f(x_1, x_2, \dots, x_5) = 2 - 1.0/120x_1x_2x_3x_4x_5 \quad (98)$$

$$\mathbf{g}(x_1, x_2, \dots, x_5) = \begin{bmatrix} -x_1 \\ -x_2 \\ -x_3 \\ -x_4 \\ -x_5 \\ x_1 - 1 \\ x_2 - 2 \\ x_3 - 3 \\ x_4 - 4 \\ x_5 - 5 \end{bmatrix} \quad (99)$$

starting at \mathbf{x}^s [2.0, 2.0, 2.0, 2.0, 2.0] (infeasible) , minimum at \mathbf{x}^* [1.0, 2.0, 3.0, 4.0, 5.0]. $f(\mathbf{x}^s) = 1.7333$ and $f(\mathbf{x}^*) = 1.0000$

- Test function 46 :

$$f(x_1, x_2, \dots, x_5) = (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6 \quad (100)$$

$$\mathbf{h}(x_1, x_2, \dots, x_5) = \begin{bmatrix} x_1^2 x_4 + \sin(x_4 - x_5) - 1 \\ x_2 + x_3^4 x_4^2 - 2 \end{bmatrix} \quad (101)$$

starting at \mathbf{x}^s [0.707107, 1.75, 0.5, 2.0, 2.0] (feasible) , minimum at \mathbf{x}^* [1.0, 1.0, 1.0, 1.0, 1.0]. $f(\mathbf{x}^s) = 3.3376$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 47 :

$$f(x_1, x_2, \dots, x_5) = (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 + (x_4 - x_5)^4 \quad (102)$$

$$\mathbf{h}(x_1, x_2, \dots, x_5) = \begin{bmatrix} x_1 + x_2^2 + x_3^3 - 3 \\ x_2 - x_3^2 + x_4 - 1 \\ x_1 x_5 - 1 \end{bmatrix} \quad (103)$$

starting at \mathbf{x}^s [2.0, 1.414214, -1.0, 0.585786, 0.5] (feasible) , minimum at \mathbf{x}^* [1.0, 1.0, 1.0, 1.0, 1.0]. $f(\mathbf{x}^s) = 12.4954$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 48 :

$$f(x_1, x_2, \dots, x_5) = (x_1 - 1)^2 + (x_2 - x_3)^2 + (x_4 - x_5)^2 \quad (104)$$

$$\mathbf{h}(x_1, x_2, \dots, x_5) = \begin{bmatrix} x_1 + x_2 + x_3 + x_4 + x_5 - 5 \\ x_3 - 2(x_4 + x_5) + 3 \end{bmatrix} \quad (105)$$

starting at \mathbf{x}^s [3.0, 5.0, -3.0, 2.0, -2.0] (feasible) , minimum at \mathbf{x}^* [1.0, 1.0, 1.0, 1.0, 1.0]. $f(\mathbf{x}^s) = 84.0000$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 49 :

$$f(x_1, x_2, \dots, x_5) = (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6 \quad (106)$$

$$\mathbf{h}(x_1, x_2, \dots, x_5) = \begin{bmatrix} x_1 + x_2 + x_3 + 4x_4 - 7 \\ x_3 + 5x_5 - 6 \end{bmatrix} \quad (107)$$

starting at \mathbf{x}^s [10.0, 7.0, 2.0, -3.0, 0.8] (feasible) , minimum at \mathbf{x}^* [1.0, 1.0, 1.0, 1.0, 1.0]. $f(\mathbf{x}^s) = 266.0001$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 50 :

$$f(x_1, x_2, \dots, x_5) = (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 + (x_4 - x_5)^4 \quad (108)$$

$$\mathbf{h}(x_1, x_2, \dots, x_5) = \begin{bmatrix} x_1 + 2x_2 + 3x_3 - 6 \\ x_2 + 2x_3 + 3x_4 - 6 \\ x_3 + 2x_4 + 3x_5 - 6 \end{bmatrix} \quad (109)$$

starting at \mathbf{x}^s [35.0, -31.0, 11.0, 5.0, -5.0] (feasible) , minimum at \mathbf{x}^* [1.0, 1.0, 1.0, 1.0, 1.0]. $f(\mathbf{x}^s) = 17416.0000$ and $f(\mathbf{x}^*) = 0.0000$

- Test function 100 :

$$f(x_1, x_2, \dots, x_7) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \quad (110)$$

$$\mathbf{g}(x_1, x_2, \dots, x_7) = \begin{bmatrix} 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \\ 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \\ 192 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \\ -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \end{bmatrix} \quad (111)$$

starting at \mathbf{x}^s [1.0, 2.0, 0.0, 4.0, 0.0, 1.0, 1.0] (feasible) , minimum at \mathbf{x}^* [2.330499, 1.951372, -0.477541, 4.365736, -0.624487, 1.038131, 1.594227]. $f(\mathbf{x}^s) = 714.0000$ and $f(\mathbf{x}^*) = 680.6297$

- Test function 113 :

$$f(x_1, x_2, \dots, x_{10}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \quad (112)$$

$$\mathbf{g}(x_1, x_2, \dots, x_{10}) = \begin{bmatrix} 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \\ -10x_1 + 8x_2 + 17x_7 - 2x_8 \\ 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \\ -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \\ -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \\ -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \\ -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \\ 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \end{bmatrix} \quad (113)$$

starting at \mathbf{x}^s [2.0, 3.0, 5.0, 5.0, 1.0, 2.0, 7.0, 3.0, 6.0, 10.0] (feasible) , minimum at \mathbf{x}^* [2.171996, 2.363683, 8.773926, 5.095984, 0.990655, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927]. $f(\mathbf{x}^s) = 753.0000$ and $f(\mathbf{x}^*) = 24.3062$

Appendix C - Performance of gradient-based algorithms on the single objective constrained test problems

This Appendix presents the complete results of the constrained gradient-based algorithms performances on the test problems given in Appendix B. The constrained algorithm parameters, which are the same for every test problem, are

- COBYLA: initial trust region $\tau_i = 0.1$, maximum function evaluations 10 000.
- LFOPC: undefined space avoidance factor $\rho_u = 1.0$. 500 iterations per phase.
- LMM-BFGS: BFGS $I_{max} = 150$, LMM It = 6, $\rho = 15.0$
- LMM-CGPR: same as *LMM-BFGS*
- LMM-LFOP: LFOP $I_{max} = 500$, LMM It = 6, $\rho = 50.0$, initial step 0.05, undefined space avoidance factor $\rho_u = 1.0$
- SQP: Hessian finite difference perturbation size $\epsilon = 0.1$, line search penalty function parameter $\rho = 100.0$
- SLSQP: $I_{max} = 1000$

Table 1 and Table 2 give a summary of the algorithm performances. The summary tables are followed by tables giving further detail on each of the algorithms' performance.

In Problem 2 and Problem 15 only the SLSQP method manages to find the function minimum, with the rest of the algorithms converging to the local minimum. Lowering the penalty factor for Problem 2 allows other methods to also find the problem minimum.

Problem 25 is an example where having a large penalty factor restricts the optimization algorithms excessively, making them unable to search the design space. Changing ρ to 1 gives the LMM method enough freedom to move around and find the solution.

The LMM-CGPR (Table 3) and LMM-BFGS (Table 4) come close to problem minimum on the majority of functions.

When examining the performance of the LFOP methods, LMM-LFOP (Table 5) and LFOPC (Table 6), the reader is reminded that these methods are designed for engineering analysis. These methods focus on robustness, not high accuracy.

The COBYLA method (Table 7) cannot handle equality constraints, hence the blank results for the COBYLA algorithm are where the test problems contained equality constraints.

The SQP (Table 8) and SLSQP (Table 9) methods are the most successful in terms of finding the function minimum. Note however that the SLSQP algorithm is far more efficient than the implemented SQP method.

| | COBYLA | LFOPC | LMM-BFGS | LMM-CGPR | LMM-LFOP | SQP | SLSQP |
|----------|--------------------|--------------------------------|---------------------------------|--------------------|-------------------|-------------------|-------------------------------|
| t_1 | 17391 _x | 1281 | 1296 | 1196 | 1856 | 885 | 78 _x |
| t_2 | - | - | - | - | - | - | 80 ^v |
| t_3 | 56 ^v | - | 582 ^v | 627 | - | 249 | - |
| t_4 | 17 ^v | 2313 | 552 ^v | 562 ^v | 6552 | 239 | 8 |
| t_5 | 47 | 486 | 291 | 321 | 476 | 335 | 26 |
| t_6 | - | - | 11881 ^v | 12296 ^v | - | 1429 ^v | 38 ^v |
| t_7 | - | - | 1732 ^v | 1732 ^v | - | 972 ^v | 42 ^v |
| t_8 | - | 556 ^v | 341 ^v | 371 ^v | 1172 ^v | 272 ^v | 18 ^v |
| t_9 | - | - | 564 ^v | 573 ^v | - | 282 ^v | 24 _x ^v |
| t_{10} | 84 ^v | 2338 _x | 3552 | 2442 ^v | - | - | 45 ^v |
| t_{11} | 73 ^v | - | 1862 ^v | 1997 ^v | - | 885 | 33 ^v |
| t_{12} | 60 ^v | 5272 | 2902 | 3138 | - | 3054 | 42 ^v |
| t_{13} | 69 ^v | 20693 ^v | - | - | - | - | 113 _x ^v |
| t_{14} | - | 1963 | 602 | 772 ^v | 1162 | 368 | 24 ^v |
| t_{15} | - | - | - | - | - | - | 21 ^v |
| t_{16} | - | 1413 _x ^v | 722 ^v | 722 ^v | - | - | - |
| t_{17} | 35 ^v | 1683 ^v | 1297 | 1462 | 2332 ^v | 511 ^v | 55 |
| t_{18} | 80 ^v | - | - | - | - | 3276 | 32 _x ^v |
| t_{19} | 35 | - | 4029 ^v | 4690 ^v | - | - | 25 ^v |
| t_{20} | - | - | 1243 ^v | 1263 ^v | - | - | 51 ^v |
| t_{21} | 55 ^v | - | 747 | 827 | - | 322 ^v | 16 ^v |
| t_{22} | 21 ^v | 1008 _x | 1572 ^v | 2699 _x | - | 365 | 26 ^v |
| t_{23} | 23 | 2078 ^v | 1197 ^v | - | 6994 | 531 | 24 |
| t_{24} | 18 ^v | 1088 _x | 893 ^v | 2307 ^v | - | - | 20 ^v |
| t_{25} | - | - | - | - | - | - | - |
| t_{26} | - | - | 11652 _x ^v | - | - | 4090 ^v | 118 _x ^v |

The number of function evaluations required to obtain the solution \mathbf{x} , with $\|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-4}$, $\max(g(\mathbf{x})) \leq 0$ and $\max(|h(\mathbf{x})|) < 10^{-4}$.

^v the constraint criteria is loosely satisfied: $0 < \max(g(\mathbf{x})) \leq 10^{-6}$ or $10^{-4} < \max|h(\mathbf{x})| < 10^{-2}$

^x the distance criteria is loosely satisfied : $10^{-4} < \|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-2}$

- the returned solution is outside the required tolerances.

Table 1: Constrained optimization algorithm performances on test functions, part A.

| | COBYLA | LFOPC | LMM-BFGS | LMM-CGPR | LMM-LFOP | SQP | SLSQP |
|-----------|------------------|--------------------------------|--------------------------------|---------------------------------|--------------------|--------------------|-------------------------------|
| t_{27} | | 5259 _x ^v | 8868 ^v | - | - | 12080 ^v | 109 _x ^v |
| t_{28} | | 2619 _x ^v | 246 ^v | - | - | 220 ^v | 21 ^v |
| t_{29} | 92 ^v | - | - | - | - | 4025 | 67 ^v |
| t_{30} | 91 ^v | 1683 _x ^v | 1494 | 25245 ^v | - | 1145 | 56 _x ^v |
| t_{31} | 89 ^v | - | 2555 | 40696 _x | - | 865 | 37 ^v |
| t_{32} | | 5302 _x | 1283 | 3728 ^v | - | 275 | 15 |
| t_{33} | 31 ^v | - | - | - | - | - | - |
| t_{34} | 40 ^v | - | - | 19131 _x | - | 8770 ^v | 40 ^v |
| t_{35} | 79 ^v | 1262 _x | 1045 ^v | 2082 _x ^v | - | 180 | 31 |
| t_{36} | 37 ^v | - | 1742 ^v | 5093 ^v | 19350 ^v | - | 10 |
| t_{37} | 96 ^v | - | - | 820 | - | - | - |
| t_{38} | - | 2494 | 2138 | 10362 _x | 2269 | 2367 | 469 |
| t_{39} | 166 ^v | - | - | - | - | - | 73 ^v |
| t_{40} | | 2090 _x ^v | 1710 ^v | - | - | - | 31 ^v |
| t_{41} | | 4061 _x | - | - | - | - | 36 _x ^v |
| t_{42} | | 4206 ^v | 2487 ^v | 7424 _x ^v | - | 2719 ^v | 39 _x ^v |
| t_{43} | 132 ^v | 2640 _x | 3825 | 24469 _x ^v | - | 3518 ^v | 62 ^v |
| t_{44} | 41 ^v | - | 2789 ^v | 14200 ^v | 24140 | - | 36 |
| t_{45} | 48 ^v | 4216 ^v | - | 9072 | - | - | 56 |
| t_{46} | | - | - | - | - | 8191 ^v | - |
| t_{47} | | 3048 _x ^v | 14503 ^v | - | - | 5016 ^v | 98 ^v |
| t_{48} | | 2861 _x ^v | 409 ^v | 470 _x ^v | - | 505 ^v | 31 ^v |
| t_{49} | | - | 1781 _x ^v | - | - | 3046 ^v | - |
| t_{50} | | 9791 _x ^v | 785 ^v | - | - | 434 ^v | 95 _x ^v |
| t_{100} | 355 ^v | - | - | - | - | 6843 ^v | 124 ^v |
| t_{113} | 337 ^v | - | - | - | - | 3312 | 147 ^v |

The number of function evaluations required to obtain the solution \mathbf{x} , with $\|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-4}$, $\max(g(\mathbf{x})) \leq 0$ and $\max(|h(\mathbf{x})|) < 10^{-4}$.

^v the constraint criteria is loosely satisfied: $0 < \max(g(\mathbf{x})) \leq 10^{-6}$ or $10^{-4} < \max|h(\mathbf{x})| < 10^{-2}$

^x the distance criteria is loosely satisfied : $10^{-4} < \|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-2}$

- the returned solution is outside the required tolerances.

Table 2: Constrained optimization algorithm performances on test functions, part B.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals |
|----|---------------------------------|-------------|--------|-----|---------------------------------|-------------|--------|
| 1 | 0 | 0 | 1196 | 27 | 0.0938 | 0 | 157854 |
| 2 | 2.4454 | 0 | 842 | 28 | 0.0996 | 0 | 140980 |
| 3 | 0 | 0 | 627 | 29 | 1.5280 | 0 | 384 |
| 4 | 0 | 0 | 562 | 30 | 0 | 0 | 25245 |
| 5 | 0 | 0 | 321 | 31 | 0.0002 | 0 | 40696 |
| 6 | 0 | 0 | 12296 | 32 | 0 | 0 | 3728 |
| 7 | 0 | 0 | 1732 | 33 | 1.5261 | 0 | 921 |
| 8 | 0 | 0 | 371 | 34 | 0.0003 | 0 | 19131 |
| 9 | 0 | 0 | 573 | 35 | 0.0063 | 0 | 2082 |
| 10 | 0 | 0 | 2442 | 36 | 0 | 0 | 5093 |
| 11 | 0.0001 | 0 | 1997 | 37 | 0 | 0 | 820 |
| 12 | 0 | 0 | 3138 | 38 | 0.0005 | 0 | 10362 |
| 13 | 0.4797 | 0 | 2169 | 39 | 3.6866 | 0 | 49643 |
| 14 | 0 | 0 | 772 | 40 | 0.0151 | 0 | 37211 |
| 15 | 3.5090 | 0 | 1594 | 41 | 1.6899 | 0 | 296 |
| 16 | 0 | 0 | 722 | 42 | 0.0037 | 0 | 7424 |
| 17 | 0 | 0 | 1462 | 43 | 0.0032 | 0 | 24469 |
| 18 | 10.3935 | 0 | 1192 | 44 | 0.0000 | 0 | 14200 |
| 19 | 0 | 0 | 4690 | 45 | 0 | 0 | 9072 |
| 20 | 0 | 0 | 1263 | 46 | 0.1234 | 0 | 131516 |
| 21 | 0 | 0 | 827 | 47 | 1.7365 | 0.0001 | 182883 |
| 22 | 0.0014 | 0 | 2699 | 48 | 0.0007 | 0 | 470 |
| 23 | 0.0000 | 0.0001 | 4760 | 49 | 0.6784 | 0 | 5581 |
| 24 | 0 | 0 | 2307 | 50 | 0.0176 | 0 | 23365 |
| 25 | 51.5606 | 0 | 88 | 100 | 1.6756 | 0 | 473 |
| 26 | 0.1621 | 0 | 250350 | 113 | 1.3244 | 0.0000 | 53533 |

[†] maximum constraint violation

Table 3: LMM-CGPR performance on test functions.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals |
|----|---------------------------------|-------------|-------|-----|---------------------------------|-------------|-------|
| 1 | 0 | 0 | 1296 | 27 | 0 | 0 | 8868 |
| 2 | 2.4454 | 0 | 822 | 28 | 0 | 0 | 246 |
| 3 | 0 | 0 | 582 | 29 | 1.5284 | 0 | 145 |
| 4 | 0 | 0 | 552 | 30 | 0 | 0 | 1494 |
| 5 | 0 | 0 | 291 | 31 | 0 | 0 | 2555 |
| 6 | 0 | 0 | 11881 | 32 | 0 | 0 | 1283 |
| 7 | 0 | 0 | 1732 | 33 | 0.8171 | 0 | 679 |
| 8 | 0 | 0 | 341 | 34 | 0.0004 | 0 | 19635 |
| 9 | 0 | 0 | 564 | 35 | 0 | 0 | 1045 |
| 10 | 0 | 0 | 3552 | 36 | 0 | 0 | 1742 |
| 11 | 0.0001 | 0 | 1862 | 37 | 10.1913 | 0 | 145 |
| 12 | 0 | 0 | 2902 | 38 | 0 | 0 | 2138 |
| 13 | 0.1744 | 0 | 4532 | 39 | 3.2388 | 0 | 2033 |
| 14 | 0 | 0 | 602 | 40 | 0 | 0 | 1710 |
| 15 | 3.5090 | 0 | 1594 | 41 | 1.6948 | 0 | 345 |
| 16 | 0 | 0 | 722 | 42 | 0 | 0 | 2487 |
| 17 | 0 | 0 | 1297 | 43 | 0 | 0 | 3825 |
| 18 | 0.1874 | 0 | 10887 | 44 | 0 | 0 | 2789 |
| 19 | 0 | 0 | 4029 | 45 | 1.8433 | 0 | 4277 |
| 20 | 0 | 0 | 1243 | 46 | 0.0578 | 0 | 7562 |
| 21 | 0 | 0 | 747 | 47 | 0 | 0 | 14503 |
| 22 | 0 | 0 | 1572 | 48 | 0 | 0 | 409 |
| 23 | 0 | 0 | 1197 | 49 | 0.0002 | 0 | 1781 |
| 24 | 0 | 0 | 893 | 50 | 0 | 0 | 785 |
| 25 | 51.5606 | 0 | 88 | 100 | 0.0422 | 0 | 9227 |
| 26 | 0.0069 | 0 | 11652 | 113 | 0.0279 | 0.0000 | 72457 |

[†] maximum constraint violation

Table 4: LMM-BFGS performance on test functions.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals |
|----|---------------------------------|-------------|-------|-----|---------------------------------|-------------|-------|
| 1 | 0.0000 | 0 | 1856 | 27 | 0.9503 | 0 | 3522 |
| 2 | 2.4454 | 0 | 3542 | 28 | 2.8644 | 0.0001 | 21021 |
| 3 | 9.9851 | 0.0217 | 15015 | 29 | 0.0343 | 0 | 21021 |
| 4 | 0 | 0 | 6552 | 30 | 0.0403 | 0 | 21021 |
| 5 | 0 | 0 | 476 | 31 | 0.0575 | 0 | 21021 |
| 6 | 2.0354 | 0 | 342 | 32 | 0.0022 | 0.0000 | 94608 |
| 7 | 0.1285 | 0 | 592 | 33 | 1.5320 | 0.0005 | 21021 |
| 8 | 0 | 0 | 1172 | 34 | 6.9110 | 0 | 17637 |
| 9 | 4.9704 | 0 | 2330 | 35 | 0.1189 | 0 | 21021 |
| 10 | 0.0712 | 0 | 1982 | 36 | 0 | 0 | 19350 |
| 11 | 1.0432 | 0 | 161 | 37 | 0.0222 | 0 | 10943 |
| 12 | 0.0471 | 0 | 15015 | 38 | 0.0000 | 0 | 2269 |
| 13 | 0.7583 | 0 | 532 | 39 | 4.6825 | 0 | 487 |
| 14 | 0 | 0 | 1162 | 40 | 0.0789 | 0 | 1874 |
| 15 | 3.5090 | 0 | 2704 | 41 | 0.2824 | 0 | 1928 |
| 16 | 1.0995 | 0 | 1448 | 42 | 0.1065 | 0.0001 | 27027 |
| 17 | 0 | 0 | 2332 | 43 | 0.0553 | 0 | 27027 |
| 18 | 11.2877 | 0 | 141 | 44 | 0.0000 | 0 | 24140 |
| 19 | 4.7759 | 0 | 1444 | 45 | 3.7428 | 0 | 266 |
| 20 | 1.0000 | 0 | 1828 | 46 | 0.6461 | 0.0000 | 85241 |
| 21 | 2.5801 | 0 | 221 | 47 | 0.8494 | 0 | 56420 |
| 22 | 1.1761 | 0 | 86 | 48 | 0.3126 | 0.0000 | 56288 |
| 23 | 0 | 0 | 6994 | 49 | 4.5745 | 0.0001 | 33033 |
| 24 | 0.1864 | 0 | 15015 | 50 | 1.1747 | 0.0000 | 33033 |
| 25 | 39.6414 | 0 | 806 | 100 | 0.2130 | 1.1130 | 45045 |
| 26 | 3.7442 | 0.0000 | 10167 | 113 | 0.3915 | 0 | 63063 |

[†] maximum constraint violation

Table 5: LMM-LFOP performance on test functions.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals |
|----|---------------------------------|-------------|-------|-----|---------------------------------|-------------|-------|
| 1 | 0.0001 | 0 | 1281 | 27 | 0.0004 | 0.0000 | 5259 |
| 2 | 2.4420 | 0 | 1488 | 28 | 0.0008 | 0 | 2619 |
| 3 | 9.9998 | 0 | 708 | 29 | 0.0134 | 0 | 10334 |
| 4 | 0.0000 | 0 | 2313 | 30 | 0.0007 | 0 | 1683 |
| 5 | 0 | 0 | 486 | 31 | 0.0220 | 0 | 3258 |
| 6 | 0.2261 | 0.0000 | 18536 | 32 | 0.0010 | 0 | 5302 |
| 7 | 0.1189 | 0 | 522 | 33 | 1.5307 | 0 | 13723 |
| 8 | 0.0000 | 0.0000 | 556 | 34 | 1.4306 | 0.0000 | 35248 |
| 9 | 0.1503 | 0 | 16062 | 35 | 0.0013 | 0 | 1262 |
| 10 | 0.0009 | 0 | 2338 | 36 | 0.2819 | 0 | 36452 |
| 11 | 1.0055 | 0 | 241 | 37 | 0.1731 | 0 | 35892 |
| 12 | 0.0001 | 0 | 5272 | 38 | 0.0000 | 0 | 2494 |
| 13 | 0 | 0 | 20693 | 39 | 2.5679 | 0 | 496 |
| 14 | 0 | 0 | 1963 | 40 | 0.0006 | 0.0000 | 2090 |
| 15 | 0.6524 | 0.1578 | 25593 | 41 | 0.0042 | 0 | 4061 |
| 16 | 0.0049 | 0 | 1413 | 42 | 0.0000 | 0 | 4206 |
| 17 | 0 | 0 | 1683 | 43 | 0.0033 | 0 | 2640 |
| 18 | 1.0436 | 0 | 25077 | 44 | 0.0000 | 0 | 13521 |
| 19 | 2.9734 | 1.7401 | 26248 | 45 | 0 | 0 | 4216 |
| 20 | 1.0000 | 0 | 22708 | 46 | 0.2036 | 0.0000 | 4500 |
| 21 | 2.5858 | 0 | 311 | 47 | 0.0002 | 0 | 3048 |
| 22 | 0.0007 | 0 | 1008 | 48 | 0.0002 | 0 | 2861 |
| 23 | 0 | 0 | 2078 | 49 | 0.3653 | 0 | 8845 |
| 24 | 0.0015 | 0 | 1088 | 50 | 0.0002 | 0 | 9791 |
| 25 | 49.3209 | 0 | 3172 | 100 | 0.1412 | 0 | 75046 |
| 26 | 0.0217 | 0 | 8128 | 113 | 0.1026 | 0 | 35661 |

[†] maximum constraint violation

Table 6: LFOPC performance on test functions.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals |
|----|---------------------------------|-------------|-------|-----|---------------------------------|-------------|-------|
| 1 | 0.0016 | 0 | 17391 | 27 | | | |
| 2 | 2.4454 | 0 | 50 | 28 | | | |
| 3 | 0 | 0 | 56 | 29 | 0 | 0 | 92 |
| 4 | 0 | 0 | 17 | 30 | 0 | 0 | 91 |
| 5 | 0 | 0 | 47 | 31 | 0 | 0 | 89 |
| 6 | | | | 32 | | | |
| 7 | | | | 33 | 0 | 0 | 31 |
| 8 | | | | 34 | 0 | 0 | 40 |
| 9 | | | | 35 | 0 | 0 | 79 |
| 10 | 0 | 0 | 84 | 36 | 0 | 0 | 37 |
| 11 | 0.0001 | 0 | 73 | 37 | 0 | 0 | 96 |
| 12 | 0 | 0 | 60 | 38 | 2.8641 | 0 | 50000 |
| 13 | 0 | 0 | 69 | 39 | 0 | 0 | 166 |
| 14 | | | | 40 | | | |
| 15 | 3.5090 | 0 | 143 | 41 | | | |
| 16 | 1.0995 | 0 | 22 | 42 | | | |
| 17 | 0 | 0 | 35 | 43 | 0 | 0 | 132 |
| 18 | 0 | 0 | 80 | 44 | 0 | 0 | 41 |
| 19 | 0 | 0 | 35 | 45 | 0 | 0 | 48 |
| 20 | 1.0000 | 0 | 22 | 46 | | | |
| 21 | 0 | 0 | 55 | 47 | | | |
| 22 | 0 | 0 | 21 | 48 | | | |
| 23 | 0 | 0 | 23 | 49 | | | |
| 24 | 0 | 0 | 18 | 50 | | | |
| 25 | 49.9910 | 0 | 50000 | 100 | 0.0000 | 0 | 355 |
| 26 | | | | 113 | 0 | 0 | 337 |

[†] maximum constraint violation

Table 7: COBYLA performance on test functions.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals |
|----|---------------------------------|-------------|-------|-----|---------------------------------|-------------|-------|
| 1 | 0 | 0 | 885 | 27 | 0.0001 | 0.0000 | 12080 |
| 2 | 2.4454 | 0 | 378 | 28 | 0 | 0 | 220 |
| 3 | 0 | 0 | 249 | 29 | 0 | 0 | 4025 |
| 4 | 0 | 0 | 239 | 30 | 0 | 0 | 1145 |
| 5 | 0 | 0 | 335 | 31 | 0 | 0 | 865 |
| 6 | 0 | 0 | 1429 | 32 | 0 | 0 | 275 |
| 7 | 0 | 0 | 972 | 33 | 1.5307 | 0.0055 | 6215 |
| 8 | 0 | 0 | 272 | 34 | 0 | 0 | 8770 |
| 9 | 0 | 0 | 282 | 35 | 0 | 0 | 180 |
| 10 | 0.0001 | 0.0002 | 14710 | 36 | 0.0542 | 0.0552 | 13210 |
| 11 | 0.0001 | 0 | 885 | 37 | 1.3555 | 0.0540 | 11360 |
| 12 | 0 | 0 | 3054 | 38 | 0 | 0 | 2367 |
| 13 | 2.6979 | 0.9019 | 5544 | 39 | 2.1590 | 0.0002 | 19500 |
| 14 | 0 | 0 | 368 | 40 | 0.0614 | 0.0003 | 19100 |
| 15 | 0.7914 | 0.2262 | 3068 | 41 | 0.0233 | 0.0001 | 484 |
| 16 | 2.6101 | 1.5000 | 103 | 42 | 0 | 0 | 2719 |
| 17 | 0.0000 | 0 | 511 | 43 | 0 | 0 | 3518 |
| 18 | 0 | 0 | 3276 | 44 | 7.0711 | 0 | 374 |
| 19 | 0.1310 | 0.0748 | 13670 | 45 | 3.8214 | 0 | 252 |
| 20 | 1.0870 | 0.0845 | 1798 | 46 | 0 | 0 | 8191 |
| 21 | 0 | 0 | 322 | 47 | 0 | 0 | 5016 |
| 22 | 0 | 0 | 365 | 48 | 0 | 0 | 505 |
| 23 | 0 | 0 | 531 | 49 | 0 | 0 | 3046 |
| 24 | 2.1582 | 0 | 156 | 50 | 0 | 0 | 434 |
| 25 | 19.9799 | 0 | 200 | 100 | 0 | 0 | 6843 |
| 26 | 0 | 0 | 4090 | 113 | 0 | 0 | 3312 |

[†] maximum constraint violation

Table 8: SQP performance on test functions.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | V^\dagger | evals |
|----|---------------------------------|-------------|-------|-----|---------------------------------|-------------|-------|
| 1 | 0.0003 | 0 | 78 | 27 | 0.0004 | 0 | 109 |
| 2 | 0 | 0 | 80 | 28 | 0 | 0 | 21 |
| 3 | 9.9996 | 0 | 12 | 29 | 0 | 0 | 67 |
| 4 | 0 | 0 | 8 | 30 | 0.0005 | 0 | 56 |
| 5 | 0.0001 | 0 | 26 | 31 | 0.0000 | 0 | 37 |
| 6 | 0 | 0 | 38 | 32 | 0 | 0 | 15 |
| 7 | 0 | 0 | 42 | 33 | 1.5307 | 0 | 25 |
| 8 | 0 | 0 | 18 | 34 | 0 | 0 | 40 |
| 9 | 0.0001 | 0 | 24 | 35 | 0 | 0 | 31 |
| 10 | 0 | 0 | 45 | 36 | 0 | 0 | 10 |
| 11 | 0.0001 | 0 | 33 | 37 | 0 | 0 | 83 |
| 12 | 0 | 0 | 42 | 38 | 0.0001 | 0 | 469 |
| 13 | 0.0006 | 0 | 113 | 39 | 0 | 0 | 73 |
| 14 | 0 | 0 | 24 | 40 | 0 | 0 | 31 |
| 15 | 0 | 0 | 21 | 41 | 0.0007 | 0 | 36 |
| 16 | 1.0995 | 0 | 30 | 42 | 0.0003 | 0 | 39 |
| 17 | 0 | 0 | 55 | 43 | 0 | 0 | 62 |
| 18 | 0.0002 | 0 | 32 | 44 | 0 | 0 | 36 |
| 19 | 0 | 0 | 25 | 45 | 0 | 0 | 56 |
| 20 | 0 | 0 | 51 | 46 | 0.1037 | 0 | 80 |
| 21 | 0 | 0 | 16 | 47 | 0.0001 | 0 | 98 |
| 22 | 0 | 0 | 26 | 48 | 0 | 0 | 31 |
| 23 | 0 | 0 | 24 | 49 | 0.2074 | 0 | 45 |
| 24 | 0 | 0 | 20 | 50 | 0.0003 | 0 | 95 |
| 25 | 51.5606 | 0 | 5 | 100 | 0.0000 | 0 | 124 |
| 26 | 0.0019 | 0 | 118 | 113 | 0.0000 | 0 | 147 |

[†] maximum constraint violation

Table 9: SLSQP performance on test functions.

Appendix D - Performance of the population-based algorithms on the single objective constrained test problems

The population-based routines performances on the constrained optimization test problems are presented in this section. The methods are stochastic, so each problem was run 11 times. The population bounds for both DE and PSO was the test problem starting point $\mathbf{x}^s \pm 0.5$.

The PSO particle count was set to 60 and the inertia factor, ω , to 0.7. The belief factors c_1 and c_2 where both set to 1.0 as done in the main text.

The DE population size was set to 60, using a difference vector strength F of 0.5 and a cross-over rate CR of 0.7. The best/1/bin scheme was used.

A heuristic is implemented for determining the number of function evaluations allocated to each testing problem. The number of function evaluations f_e , is based upon the problem dimension, n as follows:

$$f_e = \begin{cases} 5000 & \text{if } N \leq 4 \\ 10000 & \text{if } N = 5 \\ 20000 & \text{if } N \geq 6 \end{cases} \quad (114)$$

All the algorithms run until f_e as no other termination criteria are implemented.

This Appendix shows that the population based algorithm design selection criteria used successfully handles constraints, albeit with less efficiency when compared to the gradient based methods.

Tables 10 and 11 show a summary of the algorithms' performance on the test problems. This is followed by Table 12 which gives more detail on the DE optimization runs, and Table 13 which presents additional detail on the PSO runs.

Table 10: Constrained optimization algorithm performances, part A.

| | max evals. | DE | | | PSO | | |
|----------|------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| | | worst | median | best | worst | median | best |
| t_1 | 5000 | 4260 | 4080 | 3180 | - | - | 4980 _x |
| t_2 | 5000 | - | - | - | - | - | - |
| t_3 | 5000 | 4980 _x | 4980 _x | 4980 _x | - | - | - |
| t_4 | 5000 | 4560 | 3900 | 3240 | 4560 | 3780 | 3180 |
| t_5 | 5000 | 2520 | 2460 | 2220 | 4080 | 3840 | 3060 |
| t_6 | 5000 | - | - | - | - | - | - |
| t_7 | 5000 | - | - | - | - | - | - |
| t_8 | 5000 | - | 4980 _x | 4980 _x | 4980 _x | 4980 _x | 4980 _x |
| t_9 | 5000 | 4980 _x | 4980 _x | 4980 _x | - | - | - |
| t_{10} | 5000 | - | 4980 _x | 4980 _x | - | - | 4980 _x |
| t_{11} | 5000 | 4980 _x | 4980 _x | 4980 _x | - | - | 4980 _x |
| t_{12} | 5000 | 4980 _x | 4980 _x | 4980 _x | - | - | 4980 _x |
| t_{13} | 5000 | - | - | - | - | - | - |
| t_{14} | 5000 | 4980 _x | 4980 _x | 4980 _x | - | - | - |
| t_{15} | 5000 | - | - | 4920 | - | - | - |
| t_{16} | 5000 | 4980 _x | 4500 | 3900 | 4980 _x | 4980 _x | 3780 |
| t_{17} | 5000 | 4980 _x | 4980 _x | 4980 | 4980 _x | 4980 _x | 4920 |
| t_{18} | 5000 | - | 4980 _x | 4980 _x | - | - | 4980 _x |
| t_{19} | 5000 | - | - | 4980 _x | - | - | - |
| t_{20} | 5000 | - | - | 4980 _x | - | 4980 _x | 4020 |
| t_{21} | 5000 | 4980 _x | 4500 | 3480 | 4980 _x | 4980 _x | 4800 |
| t_{22} | 5000 | 4740 | 4560 | 4080 | 4500 | 4380 | 3900 |
| t_{23} | 5000 | 4980 _x | 4980 _x | 4980 _x | 4980 _x | 4980 _x | 4980 |
| t_{24} | 5000 | 4980 _x | 4440 | 3660 | 4620 | 4380 | 3840 |
| t_{25} | 5000 | - | - | - | - | - | - |
| t_{26} | 5000 | - | - | - | - | - | - |

The number of function evaluations required to obtain the solution \mathbf{x} , with $\|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-6}$, $\max(g(\mathbf{x})) \leq 0$ and $\max(|h(\mathbf{x})|) < 10^{-4}$.

^x the distance criteria is loosely satisfied : $10^{-6} < \|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-3}$

- the returned solution is outside the required tolerances.

Table 11: Constrained optimization algorithm performances on test functions.

| | max evals. | DE | | | PSO | | |
|-----------|------------|--------------------|--------------------|--------------------|-------------------|-------------------|-------------------|
| | | worst | median | best | worst | median | best |
| t_{27} | 5000 | - | - | - | - | - | - |
| t_{28} | 5000 | 4980 _x | 4980 _x | 4980 _x | - | - | - |
| t_{29} | 5000 | - | - | - | - | - | - |
| t_{30} | 5000 | 4980 _x | 4980 _x | 4980 _x | 4980 _x | 4980 _x | 4980 _x |
| t_{31} | 5000 | 4980 _x | 4980 _x | 4980 _x | - | - | - |
| t_{32} | 5000 | - | - | - | - | - | - |
| t_{33} | 5000 | - | 4980 _x | 4980 _x | 4980 _x | 4980 _x | 4980 _x |
| t_{34} | 5000 | - | - | - | - | - | 4980 _x |
| t_{35} | 5000 | 4980 _x | 4980 _x | 4980 _x | - | - | 4980 _x |
| t_{36} | 5000 | - | - | 4980 _x | - | - | 4980 _x |
| t_{37} | 5000 | - | - | - | - | - | - |
| t_{38} | 10000 | - | 9960 _x | 9960 _x | - | - | - |
| t_{39} | 10000 | - | - | 9960 _x | - | - | - |
| t_{40} | 10000 | - | - | - | - | - | - |
| t_{41} | 10000 | - | - | 9960 _x | - | - | - |
| t_{42} | 10000 | - | - | - | - | - | - |
| t_{43} | 10000 | - | - | - | - | - | - |
| t_{44} | 10000 | - | 9960 _x | 9960 _x | - | 7500 | 6720 |
| t_{45} | 20000 | 17520 | 16200 | 15300 | 19320 | 8160 | 7260 |
| t_{46} | 20000 | - | - | - | - | - | - |
| t_{47} | 20000 | - | - | - | - | - | - |
| t_{48} | 20000 | 19980 _x | 19980 _x | 19980 _x | - | - | - |
| t_{49} | 20000 | - | - | 19980 _x | - | - | - |
| t_{50} | 20000 | - | 19980 _x | 19980 _x | - | - | - |
| t_{100} | 20000 | - | - | - | - | - | - |
| t_{113} | 20000 | - | - | - | - | - | - |

The number of function evaluations required to obtain the solution \mathbf{x} , with $\|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-6}$, $\max(g(\mathbf{x})) \leq 0$ and $\max(|h(\mathbf{x})|) < 10^{-4}$.

^x the distance criteria is loosely satisfied : $10^{-6} < \|\mathbf{x} - \mathbf{x}^*\| \leq 10^{-3}$

- the returned solution is outside the required tolerances.

Table 12: DE median performance on test functions.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | $f - f^*$ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | $f - f^*$ | V^\dagger | evals |
|----|---------------------------------|-----------|-------------|-------|-----|---------------------------------|-----------|-------------|-------|
| 1 | 0 | 0 | 0 | 4080 | 27 | 0.5096 | 0.0133 | 0 | 4980 |
| 2 | 2.4454 | 4.8908 | 0 | 4980 | 28 | 0.0002 | 0 | 0 | 4980 |
| 3 | 0.0001 | 0 | 0 | 4980 | 29 | 0.0035 | 0 | 0 | 4980 |
| 4 | 0 | 0 | 0 | 3900 | 30 | 0 | 0 | 0 | 4980 |
| 5 | 0 | 0 | 0 | 2460 | 31 | 0.0004 | 0 | 0 | 4980 |
| 6 | 2.0084 | 4.0333 | 0 | 4980 | 32 | 0.0098 | 0.0005 | 0 | 4980 |
| 7 | 0.5855 | 0.4719 | 0 | 4980 | 33 | 0.0005 | 0.0005 | 0 | 4980 |
| 8 | 0 | 0 | 0 | 4980 | 34 | 0.0072 | 0.0007 | 0 | 4980 |
| 9 | 0 | 0 | 0 | 4980 | 35 | 0.0002 | 0 | 0 | 4980 |
| 10 | 0.0002 | 0 | 0 | 4980 | 36 | 0.0024 | 0.1961 | 0 | 4980 |
| 11 | 0.0001 | 0 | 0 | 4980 | 37 | 0.0133 | 0.0025 | 0 | 4980 |
| 12 | 0 | 0 | 0 | 4980 | 38 | 0 | 0 | 0 | 9960 |
| 13 | 0.0094 | 0.0189 | 0 | 4980 | 39 | 0.0018 | 0 | 0 | 9960 |
| 14 | 0 | 0 | 0 | 4980 | 40 | 0.3291 | 0.0762 | 0 | 9960 |
| 15 | 3.5090 | 53.8798 | 0 | 4980 | 41 | 0.0035 | 0 | 0 | 9960 |
| 16 | 0 | 0 | 0 | 4500 | 42 | 0.1744 | 0.1074 | 0 | 9960 |
| 17 | 0 | 0 | 0 | 4980 | 43 | 0.0029 | 0.0005 | 0 | 9960 |
| 18 | 0.0007 | 0 | 0 | 4980 | 44 | 0 | 0.0002 | 0 | 9960 |
| 19 | 0.0118 | 11.9254 | 0 | 4980 | 45 | 0 | 0 | 0 | 16200 |
| 20 | 1.0000 | 2.0001 | 0 | 4980 | 46 | 1.9305 | 5.0692 | 0 | 19980 |
| 21 | 0 | 0 | 0 | 4500 | 47 | 2.1495 | 6.2340 | 0 | 19980 |
| 22 | 0 | 0 | 0 | 4560 | 48 | 0 | 0 | 0 | 19980 |
| 23 | 0 | 0 | 0 | 4980 | 49 | 0.0020 | 0 | 0 | 19980 |
| 24 | 0 | 0 | 0 | 4440 | 50 | 0 | 0 | 0 | 19980 |
| 25 | 0.0113 | 0 | 0 | 4980 | 100 | 0.0250 | 0.0032 | 0 | 19980 |
| 26 | 3.7293 | 15.7074 | 0 | 4980 | 113 | 0.2807 | 0.4427 | 0 | 19980 |

[†] maximum constraint violation

All values below 10^{-4} are represented as 0

Table 13: PSO median performance on test functions.

| tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | $f - f^*$ | V^\dagger | evals | tf | $\ \mathbf{x} - \mathbf{x}^*\ $ | $f - f^*$ | V^\dagger | evals |
|----|---------------------------------|-----------|-------------|-------|-----|---------------------------------|-----------|-------------|-------|
| 1 | 0.0014 | 0 | 0 | 4980 | 27 | 0.6860 | 0.2180 | 0 | 4980 |
| 2 | 2.4454 | 4.8908 | 0 | 4980 | 28 | 4.5323 | 9.8031 | 0 | 4980 |
| 3 | 8.8373 | 0.0008 | 0 | 4980 | 29 | 0.0232 | 0.0026 | 0 | 4980 |
| 4 | 0 | 0 | 0 | 3780 | 30 | 0.0002 | 0 | 0 | 4980 |
| 5 | 0 | 0 | 0 | 3840 | 31 | 0.0049 | 0.0002 | 0 | 4980 |
| 6 | 1.8740 | 3.4392 | 0 | 4980 | 32 | 0.7907 | 1.5551 | 0 | 4980 |
| 7 | 0.6721 | 0.5964 | 0 | 4980 | 33 | 0 | 0 | 0 | 4980 |
| 8 | 0 | 0 | 0 | 4980 | 34 | 0.0037 | 0.0006 | 0 | 4980 |
| 9 | 4.9989 | 0.4998 | 0 | 4980 | 35 | 0.0018 | 0 | 0 | 4980 |
| 10 | 0.0066 | 0 | 0 | 4980 | 36 | 0.0017 | 0.1478 | 0 | 4980 |
| 11 | 0.0171 | 0.0004 | 0 | 4980 | 37 | 1.1089 | 9.8203 | 0 | 4980 |
| 12 | 0.0074 | 0.0002 | 0 | 4980 | 38 | 0.3147 | 0.0373 | 0 | 9960 |
| 13 | 0.0046 | 0.0093 | 0 | 4980 | 39 | 0.0155 | 0.0002 | 0 | 9960 |
| 14 | 0.3224 | 0.8084 | 0 | 4980 | 40 | 0.2086 | 0.0338 | 0 | 9960 |
| 15 | 3.5093 | 53.8802 | 0 | 4980 | 41 | 0.4492 | 0.0481 | 0 | 9960 |
| 16 | 0 | 0 | 0 | 4980 | 42 | 0.4733 | 0.7914 | 0 | 9960 |
| 17 | 0 | 0 | 0 | 4980 | 43 | 0.0612 | 0.0168 | 0 | 9960 |
| 18 | 0.0248 | 0 | 0 | 4980 | 44 | 0 | 0 | 0 | 7500 |
| 19 | 0.0621 | 62.5341 | 0 | 4980 | 45 | 0 | 0 | 0 | 8160 |
| 20 | 0 | 0.0003 | 0 | 4980 | 46 | 1.8493 | 4.8674 | 0 | 19980 |
| 21 | 0 | 0 | 0 | 4980 | 47 | 2.3710 | 9.4146 | 0 | 19980 |
| 22 | 0 | 0 | 0 | 4380 | 48 | 7.0583 | 91.1838 | 0 | 19980 |
| 23 | 0 | 0 | 0 | 4980 | 49 | 11.5855 | > 100 | 0 | 19980 |
| 24 | 0 | 0 | 0 | 4380 | 50 | 48.3291 | > 100 | 0 | 19980 |
| 25 | 48.9923 | 0.0056 | 0 | 4980 | 100 | 0.1519 | 0.2548 | 0 | 19980 |
| 26 | 3.7822 | 21.9671 | 0 | 4980 | 113 | 1.4715 | 5.3733 | 0 | 19980 |

[†] maximum constraint violation

All values below 10^{-4} are represented as 0

Appendix E - Optimization results for first airfoil multi-objective formulation

This appendix contains the results for the first airfoil multi-objective formulation. The first multi-objective formulation's objectives are the avionics box height vs. blended drag. The complete formulation is:

$$F(\mathbf{x}) = \begin{bmatrix} 3C_{D_1}(\mathbf{x}) + C_{D_2}(\mathbf{x}) + C_{D_3}(\mathbf{x}) \\ -B_h(\mathbf{x}) \end{bmatrix} \quad (115)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \text{maxLift}(\mathbf{x}) \\ B_h(\mathbf{x}) - 100\text{mm} \end{bmatrix} \quad (116)$$

$$\mathbf{b}_l = [0, 0, \dots, 0] \quad (117)$$

$$\mathbf{b}_u = [1, 1, \dots, 1] \quad (118)$$

The results presented in this Appendix are a combination of the results from all the MOEAs. The results are created by combining the MOEAs Pareto front approximations. The combined Pareto front approximation consists of:

- 0 designs from EPODE,
- 11 designs from EPOPSO,
- 42 designs from MOPSO,
- 0 designs from MOSADE.





















| thumbnail | B_h (mm) | drag ($\times 10^2$) | thumbnail | B_h (mm) | drag ($\times 10^2$) |
|---|---------------|---------------------------|--|---------------|---------------------------|
|  | 100.00 | 4.945 |  | 93.59 | 4.620 |
|  | 100.00 | 4.896 |  | 91.86 | 4.595 |
|  | 98.89 | 4.863 |  | 91.30 | 4.454 |
|  | 98.20 | 4.815 |  | 88.56 | 4.424 |
|  | 97.67 | 4.789 |  | 88.53 | 4.414 |
|  | 97.63 | 4.783 |  | 88.22 | 4.321 |
|  | 96.61 | 4.749 |  | 86.81 | 4.269 |
|  | 95.89 | 4.713 |  | 85.86 | 4.267 |
|  | 94.72 | 4.691 |  | 85.02 | 4.210 |
|  | 94.61 | 4.640 |  | 84.15 | 4.116 |

Table 14: Family of results for multi-objective formulation 1, where the avionics box height (B_h) is in mm, and the blended drag coefficients is multiplied by 100. Table (1 of 3)





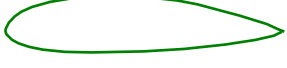















| thumbnail | B_h (mm) | drag ($\times 10^2$) | thumbnail | B_h (mm) | drag ($\times 10^2$) |
|---|---------------|---------------------------|--|---------------|---------------------------|
|  | 82.63 | 4.110 |  | 69.58 | 3.602 |
|  | 82.36 | 4.095 |  | 67.81 | 3.586 |
|  | 80.86 | 4.007 |  | 66.60 | 3.463 |
|  | 79.06 | 3.905 |  | 65.42 | 3.432 |
|  | 76.36 | 3.796 |  | 65.40 | 3.405 |
|  | 74.27 | 3.786 |  | 62.21 | 3.299 |
|  | 73.27 | 3.685 |  | 59.69 | 3.237 |
|  | 72.20 | 3.674 |  | 57.97 | 3.206 |
|  | 71.74 | 3.633 |  | 56.65 | 3.109 |
|  | 70.91 | 3.623 |  | 55.16 | 3.070 |

Table 15: Family of results for multi-objective formulation 1. Table(2 of 3)














| thumbnail | B_h (mm) | drag ($\times 10^2$) | thumbnail | B_h (mm) | drag ($\times 10^2$) |
|---|---------------|---------------------------|--|---------------|---------------------------|
|  | 53.77 | 3.048 |  | 30.13 | 2.519 |
|  | 52.08 | 3.012 |  | 28.68 | 2.474 |
|  | 49.74 | 2.999 |  | 21.29 | 2.411 |
|  | 47.50 | 2.911 | | | |
|  | 43.99 | 2.849 | | | |
|  | 41.89 | 2.814 | | | |
|  | 41.03 | 2.799 | | | |
|  | 39.02 | 2.737 | | | |
|  | 36.35 | 2.691 | | | |
|  | 33.66 | 2.621 | | | |

Table 16: Family of results for multi-objective formulation 1. Table (3 of 3)

Appendix F - Optimization results for the second air-foil multi-objective formulation

This appendix contains the results for the second multi-objective formulation. The drag coefficients are uncoupled in this formulation. The objectives are the cruise drag C_{D_1} vs. the loiter drag C_{D_2} vs. the high-speed dash drag C_{D_3} . The complete formulation is:

$$F(\mathbf{x}) = \begin{bmatrix} C_{D_1}(\mathbf{x}) \\ C_{D_2}(\mathbf{x}) \\ C_{D_3}(\mathbf{x}) \end{bmatrix} \quad (119)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \text{maxLift}(\mathbf{x}) \\ 73\text{mm}(\mathbf{x}) - B_h \end{bmatrix} \quad (120)$$

$$\mathbf{b}_l = [0, 0, \dots, 0] \quad (121)$$

$$\mathbf{b}_u = [1, 1, \dots, 1] \quad (122)$$

The results presented in this Appendix are a combination of the results from all the MOEA's. The results are created by combining the MOEAs Pareto front approximations. The combined Pareto front approximation consists of:

- 0 designs from EPODE,
- 11 designs from EPOPSO,
- 53 designs from MOPSO,
- 1 design from MOSADE.





















| thumbnail | drags ($\times 10^2$) | B_h (mm) | thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|--|---|---------------|
|  | $\begin{bmatrix} 0.682 \\ 1.001 \\ 0.767 \end{bmatrix}$ | 73.03 |  | $\begin{bmatrix} 0.691 \\ 0.940 \\ 0.642 \end{bmatrix}$ | 73.94 |
|  | $\begin{bmatrix} 0.686 \\ 0.930 \\ 0.661 \end{bmatrix}$ | 73.61 |  | $\begin{bmatrix} 0.691 \\ 0.875 \\ 0.707 \end{bmatrix}$ | 73.43 |
|  | $\begin{bmatrix} 0.687 \\ 0.925 \\ 0.662 \end{bmatrix}$ | 73.56 |  | $\begin{bmatrix} 0.691 \\ 0.911 \\ 0.648 \end{bmatrix}$ | 73.45 |
|  | $\begin{bmatrix} 0.687 \\ 1.045 \\ 0.656 \end{bmatrix}$ | 74.81 |  | $\begin{bmatrix} 0.691 \\ 0.896 \\ 0.648 \end{bmatrix}$ | 73.25 |
|  | $\begin{bmatrix} 0.688 \\ 0.966 \\ 0.659 \end{bmatrix}$ | 73.89 |  | $\begin{bmatrix} 0.693 \\ 0.889 \\ 0.679 \end{bmatrix}$ | 73.82 |
|  | $\begin{bmatrix} 0.689 \\ 0.912 \\ 0.683 \end{bmatrix}$ | 73.66 |  | $\begin{bmatrix} 0.693 \\ 0.920 \\ 0.636 \end{bmatrix}$ | 73.40 |
|  | $\begin{bmatrix} 0.689 \\ 0.957 \\ 0.658 \end{bmatrix}$ | 74.28 |  | $\begin{bmatrix} 0.693 \\ 0.895 \\ 0.677 \end{bmatrix}$ | 73.99 |
|  | $\begin{bmatrix} 0.689 \\ 0.943 \\ 0.661 \end{bmatrix}$ | 73.61 |  | $\begin{bmatrix} 0.694 \\ 0.885 \\ 0.657 \end{bmatrix}$ | 73.50 |
|  | $\begin{bmatrix} 0.690 \\ 0.932 \\ 0.658 \end{bmatrix}$ | 73.60 |  | $\begin{bmatrix} 0.699 \\ 0.869 \\ 0.692 \end{bmatrix}$ | 74.37 |
|  | $\begin{bmatrix} 0.691 \\ 0.904 \\ 0.655 \end{bmatrix}$ | 73.50 |  | $\begin{bmatrix} 0.699 \\ 0.858 \\ 0.680 \end{bmatrix}$ | 73.76 |

Table 17: Family of results for multi-objective formulation 2. The drag order is cruise, loiter and then high-speed dash. Table (1 of 4)





















| thumbnail | drags ($\times 10^2$) | B_h (mm) | thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|--|---|---------------|
|  | $\begin{bmatrix} 0.700 \\ 0.864 \\ 0.664 \end{bmatrix}$ | 73.84 |  | $\begin{bmatrix} 0.706 \\ 0.908 \\ 0.646 \end{bmatrix}$ | 73.25 |
|  | $\begin{bmatrix} 0.701 \\ 0.905 \\ 0.647 \end{bmatrix}$ | 73.17 |  | $\begin{bmatrix} 0.706 \\ 0.840 \\ 0.675 \end{bmatrix}$ | 73.48 |
|  | $\begin{bmatrix} 0.703 \\ 0.854 \\ 0.785 \end{bmatrix}$ | 73.69 |  | $\begin{bmatrix} 0.706 \\ 0.876 \\ 0.652 \end{bmatrix}$ | 73.73 |
|  | $\begin{bmatrix} 0.703 \\ 0.864 \\ 0.660 \end{bmatrix}$ | 73.50 |  | $\begin{bmatrix} 0.708 \\ 0.828 \\ 0.744 \end{bmatrix}$ | 73.47 |
|  | $\begin{bmatrix} 0.704 \\ 1.020 \\ 0.624 \end{bmatrix}$ | 74.31 |  | $\begin{bmatrix} 0.708 \\ 0.889 \\ 0.651 \end{bmatrix}$ | 73.71 |
|  | $\begin{bmatrix} 0.705 \\ 0.852 \\ 0.736 \end{bmatrix}$ | 73.24 |  | $\begin{bmatrix} 0.710 \\ 0.836 \\ 0.681 \end{bmatrix}$ | 74.01 |
|  | $\begin{bmatrix} 0.705 \\ 0.855 \\ 0.675 \end{bmatrix}$ | 73.59 |  | $\begin{bmatrix} 0.711 \\ 0.843 \\ 0.668 \end{bmatrix}$ | 73.45 |
|  | $\begin{bmatrix} 0.705 \\ 0.849 \\ 0.697 \end{bmatrix}$ | 73.47 |  | $\begin{bmatrix} 0.711 \\ 0.854 \\ 0.657 \end{bmatrix}$ | 73.69 |
|  | $\begin{bmatrix} 0.705 \\ 0.841 \\ 0.682 \end{bmatrix}$ | 74.03 |  | $\begin{bmatrix} 0.712 \\ 0.831 \\ 0.713 \end{bmatrix}$ | 74.68 |
|  | $\begin{bmatrix} 0.706 \\ 0.837 \\ 0.684 \end{bmatrix}$ | 74.21 |  | $\begin{bmatrix} 0.713 \\ 0.846 \\ 0.665 \end{bmatrix}$ | 73.73 |

Table 18: Family of results for multi-objective formulation 2. Table (2 of 4)





















| thumbnail | drags ($\times 10^2$) | B_h (mm) | thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|--|---|---------------|
|  | $\begin{bmatrix} 0.713 \\ 0.823 \\ 0.696 \end{bmatrix}$ | 73.87 |  | $\begin{bmatrix} 0.727 \\ 0.818 \\ 0.688 \end{bmatrix}$ | 73.79 |
|  | $\begin{bmatrix} 0.714 \\ 0.834 \\ 0.668 \end{bmatrix}$ | 73.43 |  | $\begin{bmatrix} 0.727 \\ 0.812 \\ 0.696 \end{bmatrix}$ | 73.38 |
|  | $\begin{bmatrix} 0.714 \\ 0.833 \\ 0.678 \end{bmatrix}$ | 73.94 |  | $\begin{bmatrix} 0.728 \\ 0.811 \\ 0.710 \end{bmatrix}$ | 73.34 |
|  | $\begin{bmatrix} 0.718 \\ 0.829 \\ 0.684 \end{bmatrix}$ | 73.95 |  | $\begin{bmatrix} 0.732 \\ 0.817 \\ 0.681 \end{bmatrix}$ | 73.09 |
|  | $\begin{bmatrix} 0.719 \\ 0.825 \\ 0.692 \end{bmatrix}$ | 73.89 |  | $\begin{bmatrix} 0.734 \\ 0.807 \\ 0.703 \end{bmatrix}$ | 73.30 |
|  | $\begin{bmatrix} 0.720 \\ 0.824 \\ 0.690 \end{bmatrix}$ | 73.99 |  | $\begin{bmatrix} 0.736 \\ 0.812 \\ 0.701 \end{bmatrix}$ | 73.59 |
|  | $\begin{bmatrix} 0.722 \\ 0.820 \\ 0.719 \end{bmatrix}$ | 73.80 |  | $\begin{bmatrix} 0.737 \\ 1.019 \\ 0.622 \end{bmatrix}$ | 73.72 |
|  | $\begin{bmatrix} 0.722 \\ 0.827 \\ 0.684 \end{bmatrix}$ | 73.95 |  | $\begin{bmatrix} 0.738 \\ 0.811 \\ 0.688 \end{bmatrix}$ | 73.30 |
|  | $\begin{bmatrix} 0.723 \\ 1.017 \\ 0.628 \end{bmatrix}$ | 73.19 |  | $\begin{bmatrix} 0.744 \\ 0.989 \\ 0.627 \end{bmatrix}$ | 73.50 |
|  | $\begin{bmatrix} 0.724 \\ 0.819 \\ 0.683 \end{bmatrix}$ | 73.15 |  | $\begin{bmatrix} 0.744 \\ 0.805 \\ 0.717 \end{bmatrix}$ | 73.52 |

Table 19: Family of results for multi-objective formulation 2. Table (3 of 4)






| thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|
|  | $\begin{bmatrix} 0.746 \\ 0.805 \\ 0.697 \end{bmatrix}$ | 73.32 |
|  | $\begin{bmatrix} 0.749 \\ 0.804 \\ 0.722 \end{bmatrix}$ | 73.62 |
|  | $\begin{bmatrix} 0.749 \\ 0.804 \\ 0.702 \end{bmatrix}$ | 73.41 |
|  | $\begin{bmatrix} 0.877 \\ 0.515 \\ 0.824 \end{bmatrix}$ | 74.55 |
|  | $\begin{bmatrix} 0.896 \\ 1.200 \\ 0.613 \end{bmatrix}$ | 73.70 |

Table 20: Family of results for multi-objective formulation 2. Table (4 of 4)

Appendix G - Optimization results for the third airfoil multi-objective formulation

This appendix contains the results for the third multi-objective formulation which has four objectives. The objectives are the cruise drag C_{D_1} vs. the loiter drag C_{D_2} vs. the high-speed dash drag C_{D_3} and the avionics box height. The complete formulation is:

$$F(\mathbf{x}) = \begin{bmatrix} C_{D_1}(\mathbf{x}) \\ C_{D_2}(\mathbf{x}) \\ C_{D_3}(\mathbf{x}) \\ -B_h(\mathbf{x}) \end{bmatrix} \quad (123)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 0.75 - \max\text{Lift}(\mathbf{x}) \\ B_h(\mathbf{x}) - 100\text{mm} \end{bmatrix} \quad (124)$$

$$\mathbf{b}_l = [0, 0, \dots, 0] \quad (125)$$

$$\mathbf{b}_u = [1, 1, \dots, 1] \quad (126)$$

The results presented in this Appendix are a combination of the results from all the MOEA's. The results are created by combining the MOEAs Pareto front approximations. The combined Pareto front approximation consists of:

- 32 designs from EPODE,
- 45 designs from EPOPSO,
- 17 designs from MOPSO,
- 4 designs from MOSADE.

It should be noted that designs are not evenly distributed according to the avionics box height, with a large proportion having avionics box height close to 100 mm.

Some of the thinner airfoils are also suspicious, appearing to be aerodynamically infeasible.





















| thumbnail | drags ($\times 10^2$) | B_h (mm) | thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|--|---|---------------|
|  | $\begin{bmatrix} 0.386 \\ 0.771 \\ 1.005 \end{bmatrix}$ | 10.91 |  | $\begin{bmatrix} 0.736 \\ 0.640 \\ 0.766 \end{bmatrix}$ | 36.52 |
|  | $\begin{bmatrix} 0.829 \\ 0.797 \\ 0.370 \end{bmatrix}$ | 20.19 |  | $\begin{bmatrix} 0.605 \\ 0.861 \\ 0.516 \end{bmatrix}$ | 38.06 |
|  | $\begin{bmatrix} 0.480 \\ 0.700 \\ 0.663 \end{bmatrix}$ | 22.84 |  | $\begin{bmatrix} 0.568 \\ 0.673 \\ 0.596 \end{bmatrix}$ | 38.99 |
|  | $\begin{bmatrix} 0.486 \\ 0.697 \\ 0.429 \end{bmatrix}$ | 24.25 |  | $\begin{bmatrix} 0.870 \\ 0.795 \\ 0.818 \end{bmatrix}$ | 39.36 |
|  | $\begin{bmatrix} 0.527 \\ 0.575 \\ 0.591 \end{bmatrix}$ | 24.88 |  | $\begin{bmatrix} 0.703 \\ 0.705 \\ 1.136 \end{bmatrix}$ | 40.08 |
|  | $\begin{bmatrix} 0.740 \\ 0.565 \\ 0.751 \end{bmatrix}$ | 25.54 |  | $\begin{bmatrix} 1.057 \\ 1.121 \\ 0.510 \end{bmatrix}$ | 40.28 |
|  | $\begin{bmatrix} 0.542 \\ 0.571 \\ 0.802 \end{bmatrix}$ | 26.52 |  | $\begin{bmatrix} 1.193 \\ 0.707 \\ 1.163 \end{bmatrix}$ | 40.52 |
|  | $\begin{bmatrix} 0.523 \\ 0.723 \\ 0.713 \end{bmatrix}$ | 29.60 |  | $\begin{bmatrix} 0.632 \\ 0.809 \\ 0.835 \end{bmatrix}$ | 43.22 |
|  | $\begin{bmatrix} 0.534 \\ 0.610 \\ 0.479 \end{bmatrix}$ | 33.05 |  | $\begin{bmatrix} 0.592 \\ 1.252 \\ 0.845 \end{bmatrix}$ | 43.80 |
|  | $\begin{bmatrix} 0.567 \\ 1.042 \\ 0.474 \end{bmatrix}$ | 34.51 |  | $\begin{bmatrix} 0.605 \\ 0.790 \\ 0.858 \end{bmatrix}$ | 46.69 |

Table 21: Family of results for multi-objective formulation 3. The drag order is cruise, loiter and then high-speed dash. Table (1 of 5)





















| thumbnail | drags ($\times 10^2$) | B_h (mm) | thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|--|---|---------------|
|  | $\begin{bmatrix} 0.637 \\ 0.894 \\ 0.523 \end{bmatrix}$ | 47.57 |  | $\begin{bmatrix} 1.003 \\ 1.323 \\ 0.731 \end{bmatrix}$ | 78.52 |
|  | $\begin{bmatrix} 1.035 \\ 0.728 \\ 0.620 \end{bmatrix}$ | 47.96 |  | $\begin{bmatrix} 0.888 \\ 0.968 \\ 1.059 \end{bmatrix}$ | 79.65 |
|  | $\begin{bmatrix} 0.639 \\ 0.831 \\ 0.572 \end{bmatrix}$ | 50.64 |  | $\begin{bmatrix} 0.805 \\ 2.105 \\ 0.762 \end{bmatrix}$ | 80.41 |
|  | $\begin{bmatrix} 0.659 \\ 0.800 \\ 0.648 \end{bmatrix}$ | 53.29 |  | $\begin{bmatrix} 0.836 \\ 1.349 \\ 0.753 \end{bmatrix}$ | 80.64 |
|  | $\begin{bmatrix} 0.614 \\ 0.831 \\ 0.907 \end{bmatrix}$ | 54.48 |  | $\begin{bmatrix} 1.347 \\ 1.949 \\ 0.702 \end{bmatrix}$ | 80.70 |
|  | $\begin{bmatrix} 0.802 \\ 1.282 \\ 0.615 \end{bmatrix}$ | 54.61 |  | $\begin{bmatrix} 1.343 \\ 1.790 \\ 0.729 \end{bmatrix}$ | 81.41 |
|  | $\begin{bmatrix} 0.890 \\ 0.725 \\ 0.843 \end{bmatrix}$ | 54.84 |  | $\begin{bmatrix} 0.801 \\ 2.031 \\ 0.884 \end{bmatrix}$ | 82.04 |
|  | $\begin{bmatrix} 0.683 \\ 0.942 \\ 0.637 \end{bmatrix}$ | 60.60 |  | $\begin{bmatrix} 0.864 \\ 1.992 \\ 0.751 \end{bmatrix}$ | 82.73 |
|  | $\begin{bmatrix} 0.703 \\ 1.510 \\ 0.635 \end{bmatrix}$ | 60.80 |  | $\begin{bmatrix} 0.874 \\ 2.485 \\ 0.770 \end{bmatrix}$ | 82.87 |
|  | $\begin{bmatrix} 0.719 \\ 1.494 \\ 0.653 \end{bmatrix}$ | 61.26 |  | $\begin{bmatrix} 0.899 \\ 0.972 \\ 1.097 \end{bmatrix}$ | 83.81 |

Table 22: Family of results for multi-objective formulation 3. Table (2 of 5)





















| thumbnail | drags ($\times 10^2$) | B_h (mm) | thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|--|---|---------------|
|  | $\begin{bmatrix} 1.003 \\ 1.323 \\ 0.731 \end{bmatrix}$ | 78.52 |  | $\begin{bmatrix} 0.948 \\ 1.255 \\ 0.753 \end{bmatrix}$ | 84.84 |
|  | $\begin{bmatrix} 0.888 \\ 0.968 \\ 1.059 \end{bmatrix}$ | 79.65 |  | $\begin{bmatrix} 0.847 \\ 1.183 \\ 0.937 \end{bmatrix}$ | 84.95 |
|  | $\begin{bmatrix} 0.805 \\ 2.105 \\ 0.762 \end{bmatrix}$ | 80.41 |  | $\begin{bmatrix} 1.368 \\ 1.806 \\ 0.771 \end{bmatrix}$ | 85.09 |
|  | $\begin{bmatrix} 0.836 \\ 1.349 \\ 0.753 \end{bmatrix}$ | 80.64 |  | $\begin{bmatrix} 0.879 \\ 0.985 \\ 0.824 \end{bmatrix}$ | 86.08 |
|  | $\begin{bmatrix} 1.347 \\ 1.949 \\ 0.702 \end{bmatrix}$ | 80.70 |  | $\begin{bmatrix} 0.861 \\ 1.046 \\ 0.835 \end{bmatrix}$ | 86.74 |
|  | $\begin{bmatrix} 1.343 \\ 1.790 \\ 0.729 \end{bmatrix}$ | 81.41 |  | $\begin{bmatrix} 1.264 \\ 0.679 \\ 1.211 \end{bmatrix}$ | 87.86 |
|  | $\begin{bmatrix} 0.801 \\ 2.031 \\ 0.884 \end{bmatrix}$ | 82.04 |  | $\begin{bmatrix} 1.457 \\ 1.949 \\ 0.774 \end{bmatrix}$ | 88.03 |
|  | $\begin{bmatrix} 0.864 \\ 1.992 \\ 0.751 \end{bmatrix}$ | 82.73 |  | $\begin{bmatrix} 1.144 \\ 0.709 \\ 1.123 \end{bmatrix}$ | 88.38 |
|  | $\begin{bmatrix} 0.874 \\ 2.485 \\ 0.770 \end{bmatrix}$ | 82.87 |  | $\begin{bmatrix} 0.890 \\ 2.505 \\ 0.889 \end{bmatrix}$ | 88.49 |
|  | $\begin{bmatrix} 0.899 \\ 0.972 \\ 1.097 \end{bmatrix}$ | 83.81 |  | $\begin{bmatrix} 0.921 \\ 1.035 \\ 0.860 \end{bmatrix}$ | 89.43 |

Table 23: Family of results for multi-objective formulation 3. Table (3 of 5)





















| thumbnail | drags ($\times 10^2$) | B_h (mm) | thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|--|---|---------------|
|  | $\begin{bmatrix} 1.642 \\ 2.036 \\ 0.810 \end{bmatrix}$ | 89.51 |  | $\begin{bmatrix} 0.918 \\ 1.536 \\ 0.862 \end{bmatrix}$ | 93.85 |
|  | $\begin{bmatrix} 0.940 \\ 2.255 \\ 0.833 \end{bmatrix}$ | 90.25 |  | $\begin{bmatrix} 0.922 \\ 1.801 \\ 0.882 \end{bmatrix}$ | 94.62 |
|  | $\begin{bmatrix} 0.892 \\ 1.117 \\ 0.941 \end{bmatrix}$ | 90.46 |  | $\begin{bmatrix} 0.927 \\ 1.391 \\ 0.844 \end{bmatrix}$ | 96.81 |
|  | $\begin{bmatrix} 0.938 \\ 1.026 \\ 1.051 \end{bmatrix}$ | 91.37 |  | $\begin{bmatrix} 0.948 \\ 1.214 \\ 0.872 \end{bmatrix}$ | 97.14 |
|  | $\begin{bmatrix} 0.969 \\ 1.116 \\ 1.200 \end{bmatrix}$ | 91.51 |  | $\begin{bmatrix} 1.117 \\ 0.776 \\ 1.212 \end{bmatrix}$ | 97.36 |
|  | $\begin{bmatrix} 0.880 \\ 1.698 \\ 1.074 \end{bmatrix}$ | 91.61 |  | $\begin{bmatrix} 0.970 \\ 1.971 \\ 0.893 \end{bmatrix}$ | 97.48 |
|  | $\begin{bmatrix} 1.423 \\ 2.131 \\ 0.793 \end{bmatrix}$ | 92.21 |  | $\begin{bmatrix} 1.005 \\ 3.276 \\ 0.873 \end{bmatrix}$ | 97.51 |
|  | $\begin{bmatrix} 1.122 \\ 1.652 \\ 0.829 \end{bmatrix}$ | 92.72 |  | $\begin{bmatrix} 0.528 \\ 1.135 \\ 1.084 \end{bmatrix}$ | 98.52 |
|  | $\begin{bmatrix} 0.900 \\ 2.604 \\ 0.871 \end{bmatrix}$ | 92.90 |  | $\begin{bmatrix} 1.159 \\ 1.669 \\ 0.920 \end{bmatrix}$ | 98.68 |
|  | $\begin{bmatrix} 1.592 \\ 2.265 \\ 0.799 \end{bmatrix}$ | 93.40 |  | $\begin{bmatrix} 1.008 \\ 1.134 \\ 1.125 \end{bmatrix}$ | 98.98 |

Table 24: Family of results for multi-objective formulation 3. Table (4 of 5)



















| thumbnail | drags ($\times 10^2$) | B_h (mm) | thumbnail | drags ($\times 10^2$) | B_h (mm) |
|---|---|---------------|--|---|---------------|
|  | $\begin{bmatrix} 0.993 \\ 2.363 \\ 0.875 \end{bmatrix}$ | 99.10 |  | $\begin{bmatrix} 1.708 \\ 1.973 \\ 1.299 \end{bmatrix}$ | 99.87 |
|  | $\begin{bmatrix} 0.998 \\ 1.138 \\ 0.964 \end{bmatrix}$ | 99.12 |  | $\begin{bmatrix} 1.409 \\ 5.548 \\ 1.288 \end{bmatrix}$ | 99.90 |
|  | $\begin{bmatrix} 1.044 \\ 3.353 \\ 0.846 \end{bmatrix}$ | 99.23 |  | $\begin{bmatrix} 1.921 \\ 2.378 \\ 0.894 \end{bmatrix}$ | 99.94 |
|  | $\begin{bmatrix} 2.533 \\ 3.351 \\ 0.863 \end{bmatrix}$ | 99.40 |  | $\begin{bmatrix} 1.506 \\ 2.206 \\ 1.030 \end{bmatrix}$ | 99.94 |
|  | $\begin{bmatrix} 0.979 \\ 2.103 \\ 0.900 \end{bmatrix}$ | 99.46 |  | $\begin{bmatrix} 1.815 \\ 2.899 \\ 1.129 \end{bmatrix}$ | 99.95 |
|  | $\begin{bmatrix} 1.674 \\ 2.134 \\ 0.897 \end{bmatrix}$ | 99.68 |  | $\begin{bmatrix} 1.030 \\ 1.498 \\ 1.489 \end{bmatrix}$ | 99.99 |
|  | $\begin{bmatrix} 1.041 \\ 1.188 \\ 0.980 \end{bmatrix}$ | 99.70 |  | $\begin{bmatrix} 2.265 \\ 2.722 \\ 1.487 \end{bmatrix}$ | 100.00 |
|  | $\begin{bmatrix} 1.384 \\ 1.861 \\ 0.904 \end{bmatrix}$ | 99.72 |  | $\begin{bmatrix} 2.287 \\ 2.438 \\ 2.211 \end{bmatrix}$ | 100.00 |
|  | $\begin{bmatrix} 1.354 \\ 1.496 \\ 1.260 \end{bmatrix}$ | 99.85 | | | |
|  | $\begin{bmatrix} 2.641 \\ 1.917 \\ 1.049 \end{bmatrix}$ | 99.87 | | | |

Table 25: Family of results for multi-objective formulation 3. Table (5 of 5)