

Cranfield University

Bo Xu

**A New Method Using B-splines as Shape Functions
and the Knot Insertion Algorithm for Shape
Optimization**

PhD thesis

School of Engineering

ProQuest Number: 10832268

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10832268

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by Cranfield University.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Cranfield University, School of Engineering
Department of Process and Systems Engineering
Applied Mathematics and Computing Group

PhD thesis

Academic Year 2006-2007

Bo Xu

**A New Method Using B-splines as Shape Functions
and the Knot Insertion Algorithm for Shape
Optimization**

Supervisors: Dr. Peter A. Sherar

Professor Chris Thompson

September 2007

The thesis is submitted in fulfilment of the requirement for the degree of
Doctor of Philosophy

©Cranfield University 2007. All rights reserved. No part of this publication
may be reproduced without the written permission of the copyright owner.

Abstract

A new method is developed to deal with shape optimization problems. The core idea of the method is to introduce the knot insertion algorithm which keeps the geometry unchanged while increasing the number of control points. In addition to this idea, the super-reduced idea and the mesh refinement are also employed to deal with the equality constrained optimization problem.

The developed method has been tested on several applications. The first application is a Poisson equation problem. The result produced by the new method is compared with the result produced by the BFGS method because the BFGS method is considered to be one of the best methods in optimization. The result shows that the new method is more efficient than the BFGS method.

The second application of the method is an airfoil design problem. The performance of using the new method is compared with the performance of using the EXTREM method in two design cases, RAE 2822 and NACA 0012. In both of these cases, the new method is much more efficient than the EXTREM method. The 3-dimensional tested case used is a mathematical problem. In this case, one finds that the discretization error is much bigger when compared with the 2-dimensional case. However the method still converges quickly to the optimum solution.

As well as the above applications, the use of high order B-splines and multi-

objective optimization have also been investigated. In summary, a new method is developed for shape optimization problems and validation has been carried out on several numerical examples. With the idea of the new method, it is possible to improve the efficiency of the method currently used as long as B-splines are used to describe the geometry.

Acknowledgements

First and foremost, I would like to express my profound gratitude and respect to my supervisors, Dr. Peter A. Sherar and Professor Chris P. Thompson, for their valuable guidance, suggestions, and encouragement throughout my postgraduate study at Cranfield University. This thesis could not have been completed without their support.

I am also indebted to Dr. Bowen Zhong for his helpful comments, instruction on papers and the code to compute the airfoil design problem.

I would also like to thank my wife, L. Song and my parents for the moral support they provided during the course of this Ph.D. Without them this work would have not been possible. I must also mention the help received from Dulceneia Becker, Aaron Latty and Rachael Wiseman.

Finally, I thankfully acknowledge the financial support provided by the Applied Mathematics and Computing Group at Cranfield University and the Overseas Student Research Award during the course of this Ph.D.

Contents

1	Introduction	1
2	Literature review	12
2.1	Introduction	12
2.2	Optimization theory	16
2.2.1	Unconstrained optimization	17
2.2.2	Line search algorithm and trust region method	22
2.2.3	Constrained optimization	27
2.3	Computing gradient in shape optimization	36
2.3.1	Finite difference approach	37
2.3.2	Sensitivity analysis	37
2.3.3	Adjoint method	38
2.3.4	Automatic differentiation	40
2.3.5	Hybrid method	43
2.4	Shape parameterization	44
2.5	PDE solution	48
2.5.1	The mesh	49
2.5.2	Finite difference method	53
2.5.3	Finite Element Method	55
2.5.4	Subspace method	57
3	The Super-Reduced method	59

3.1	Knot insertion algorithm	62
3.1.1	Use of knot insertion algorithm	64
3.2	Mesh refinement	66
3.3	Super-reduced idea to compute gradients	67
3.4	The procedure for the new method	69
3.5	The advantages for the new method	70
3.6	Preventing self-intersecting shapes	71
3.7	Algorithms	73
3.8	The choice of the end condition	76
3.9	Additional comments on the new method	78
3.10	Summary	79
4	Numerical tests for the method in 2-dimensional cases	81
4.1	Application for using Poisson's equation as the governing equation	82
4.1.1	Technical requirements	83
4.1.2	Theoretical results in shape representation	85
4.1.3	Compute the super-reduced gradient by using the finite difference method	88
4.1.4	Optimum shapes	89
4.1.5	CPU time	94
4.2	Results for a new objective function	96
4.2.1	Description	96
4.2.2	Compute the objective function	97
4.2.3	Results	97

4.3	Comparing with BFGS	99
4.4	Error analysis	102
4.4.1	Analysis of the influence of triangulation error	103
4.4.2	Analysis of the influence of least square error	105
4.4.3	Conclusion on error analysis	106
4.5	The control points for the initial shapes	106
4.6	Using high order B-splines	110
4.6.1	Issues with the knot insertion algorithm	111
4.7	Application to airfoil design	113
4.7.1	The geometry representation	114
4.7.2	The flow solver and the optimization algorithm	115
4.7.3	RAE 2822	116
4.7.4	NACA 0012	118
4.8	Performance comparison for methods in the design case based on NACA 0012	121
4.9	Multiobjective Optimization Design of Transonic Airfoils	123
4.9.1	The multiobjective optimization method	125
4.9.2	Results for multiobjective optimization problems	126
4.10	A new analysis for Poisson's problem	129
4.10.1	A new theoretical results in shape representation	130
4.10.2	Computation of the the area with variable interval length	131
4.10.3	Compute the super-reduced gradient by using the finite difference method	134
4.11	Summary	136

5	Numerical experiments in 3-dimensional case	139
5.1	Application for using Poisson's equation as the governing equation	139
5.2	Solution Technique	140
5.3	Theoretical results in shape representation	141
5.4	Mesh considerations	144
5.5	Other methods of solution	146
5.5.1	Penalty coefficient	146
5.5.2	Lagrange-based method	147
5.6	The developed method used in a 3-dimensional case	149
5.6.1	CPU time	150
5.7	Error estimation	151
5.7.1	Analysis of the influence of the residual error	152
5.7.2	Analysis of the influence of triangulation error	154
5.7.3	Conclusion on error analysis	155
5.8	Results	155
5.9	Summary	157
6	Conclusion and Future work	159
A	Publications during the Ph.D study	174
B	Mathematical justification for super-reduced-method	174
C	Solution to the Poisson equation	177
D	Data collection with mesh generator BAMG	178

E	The second best paper award paper: An Optimization Method Based On B-spline Shape Functions & the Knot Insertion Al- gorithm	179
F	An example of using adjoint method	191

List of Figures

1	The null-space of A	30
2	Typical procedure for solving shape optimization problems, which includes geometry representation, mesh generation, PDE solution, and optimization scheme	84
3	Sample splines to construct the closed geometry. Both of the upper curve and the lower curve are splines	85
4	Initial shape 1	90
5	Optimum shape for initial shape 1 without using mesh refine- ment. Coarse mesh is in use with about 3300 elements	90
6	Optimum shape for initial shape 1 with $\ g\ < 7.0e - 3$	90
7	Optimum shape for initial shape 1 with $\ g\ < 4.5e - 3$	90
8	Optimum shape for initial shape 1 with $\ g\ < 3.5e - 3$	91
9	Optimum shape for initial shape 1 with $\ g\ < 2.5e - 3$	91
10	Initial shape 2	91
11	Optimum shape for initial shape 2 with $\ g\ < 4.5e - 3$	91
12	Initial shape 3	91
13	Optimum shape for initial shape 3 with $\ g\ < 3.5e - 3$	91
14	Optimum shape for initial shape 1 by using the new method. The knot insertion algorithm has been employed. The final number of control points are 40	93
15	Optimum shape for initial shape 3 by using the new method. The knot insertion algorithm has been employed. The final number of control points are 40	93

16	Optimum shape for initial shape 1 with $\ g\ < 3.0e - 3$	98
17	Optimum shape for initial shape 1 with $\ g\ < 2.0e - 3$	98
18	Optimum shape for initial shape 1 with $\ g\ < 1.5e - 3$	98
19	Optimum shape for initial shape 3 with $\ g\ < 1.5e - 3$	98
20	Performance comparison with BFGS the method (without using mesh refinement). The BFGS method used in this case does not use the mesh refinement. The purpose of doing so is to show how important mesh refinement is in geometry optimization	101
21	Performance comparison with the BFGS method (using mesh refinement)	101
22	Initial and optimal geometry for RAE 2822. The picture shows that the geometry constraint is kept unchanged	117
23	Initial and optimal pressure distribution for RAE 2822, The picture shows that the shock wave has been decreased	117
24	Initial and optimal geometry for NACA 0012. The picture shows that the geometry constraint is kept unchanged	120
25	Initial and optimal pressure distribution for NACA 0012. The picture shows that the shock wave has been decreased	120
26	Comparison with the BFGS method in the NACA 0012 design case	122
27	The comparison of drag coefficients for initial airfoil and optimal airfoil (NACA 0012)	128
28	The comparison of pressure distribution for initial airfoil and optimal airfoil (NACA 0012)	128

29	The comparison of geometries for initial airfoil and optimal airfoil (NACA 0012)	128
30	The comparison of drag coefficients for initial airfoil and optimal airfoil (RAE 2822)	128
31	The comparison of pressure distribution for initial airfoil and optimal airfoil (RAE 2822)	128
32	The comparison of geometries for initial airfoil and optimal airfoil (RAE 2822)	128
33	Initial shapes (top view)	156
34	Initial shapes (bottom view)	156
35	Optimal shapes (top view)	157
36	Optimal shapes (bottom view)	157
37	Optimal shapes (top view)	157
38	Optimal shapes (bottom view)	157
39	Optimal shapes (top view)	157
40	Optimal shapes (bottom view)	157
41	Initial shape	187
42	optimum shape produced with the new method	187
43	Convergence performance comparison with BFGS method . .	188
44	Initial and optimal pressure distribution for NACA 0012 . . .	190
45	The comparison of drag coefficients for initial airfoil and optimal airfoil (NACA 0012)	190

List of Tables

1	2-dimensiional shape optimization using mesh refinement . . .	90
2	Comparison for 2-dimensional shape optimization using the knot insertion algorithm and not using the knot insertion al- gorithm	92
3	CPU time for the optimization scheme in 2-dimensional case .	95
4	CPU time for File I/O in 2-dimensional case	95
5	Optimization results for the new objective function in 2-dimensional case	98
6	Performance comparison with BFGS method (without using mesh refinement)	99
7	Performance comparison with BFGS method (using mesh re- finement)	100
8	CPU time spend on BFGS method and the new method . . .	100
9	Triangulation error for different number of nodes in 2-dimensional case	104
10	Error (triangluation error plus least square error) for different number of nodes in 2-dimensional case	105
11	Value of control points for different initial shapes in 2-dimensional case	108
12	Efficiency investigation for using high order B-splines (case 1)	112
13	Efficiency investigation for using high order B-splines (case 2)	112
14	Performance comparison for the two methods in the design case of RAE 2822	117

15 Performance comparison for the two methods in the design
case of NACA 0012 119

16 Performance comparison for the three methods 122

17 CPU time distribution in 3-dimensional case 151

18 File I/O time in 3-dimensional case 151

19 Residual error investigation in 3-dimensional case 153

20 Triangulation error investigation in 3-dimensional case 154

21 Triangulation error investigation in 3-dimensional case (func-
tional B-spline surface case) 155

22 Discretization error analysis for the solution of Poisson’s equa-
tion 188

23 Performance comparison of the new method and the EXTREM
method 190

Nomenclature

A, C	general matrix
c_i	the i -th constraint equation
$\mathbf{x}_0$	initial point
$\mathbf{x}_k$	k -th iteration point
$\mathbf{x}^*$	extremal value
$\mathbf{g}$	gradient vector
$\mathbf{s}$	search direction
H	Hessian matrix
λ_i	the i -th Lagrange Multiplier
L	Lagrangian function
Ω	spatial domain
$\partial\Omega$	boundary of spatial domain
F	objective function, $\mathbb{R}^n \rightarrow \mathbb{R}$
f	objective function, $\mathbb{R} \rightarrow \mathbb{R}$
F^ϵ	the value of objective function after small perturbation
$\mathbb{R}^n$	n -dimensional real space
J	Jacobian matrix
B^{DFP}	DFP update of the inverse of Hessian matrix
B^{BFGS}	BFGS update of the inverse of Hessian matrix
d	control point in B-splines (B-spline curves and B-spline surfaces)
V	Volume
Γ^U	the upper geometry of the boundary

Γ^L	the lower geometry of the boundary
Φ	state variable, real number
$\mathbf{U}$	state variables, vector
α_k	step size in k -th optimization, or the k -th design variable
β, ρ, θ	coefficient, real number

1 Introduction

Shape optimization deals with the problems of finding a shape which is optimal in a certain sense, i.e. finding a shape which can minimize a certain objective function with or without constraints. It has been used widely in structural mechanics, fluid mechanics and even noise reduction.

Industrial applications can be found for shape optimization, such as: drag reduction for airplanes, (finding a shape that minimizes the drag), the design of a harbour which minimizes the effect of waves coming from afar. In shape optimization, the problem is posed as a minimization problem of a certain objective function (or a multi-objective function) with or without constraints. Therefore a proper optimization scheme is needed to solve this kind of problem. In general the method for solving the shape optimization problems in CFD (Computational Fluid Dynamics) can be distinguished into two types:

- Local methods
- Global methods

Local methods use the information supplied by gradients to look for the local minimum. Thus it is obvious that local methods have the limitation of finding only one of multiple minima in the case that there are many. In spite of this disadvantage the method is very efficient. It has the efficiency of finding

one local optimum with a finite number of iterations.

Global methods, such as the methods based on genetic algorithms, are aimed at finding the global optimum. These methods are particularly useful when many local optima exist for a given problem.

A hybrid method is therefore presented in order to benefit from the efficiency of local methods and the global priority of the genetic algorithm. Because the hybrid method employs techniques used in the local methods, this method can decrease the computational cost compared with the purely global method. However, comparing with the gradient-based method, the computational cost is still much more expensive because the large amount of function calls to evaluate the “fitness”, another word for the value of the objective function used in genetic algorithms.

The important issue when using gradient-based methods to solve a general optimization problem is the computation of the gradient. However, for a shape optimization problem in CFD, which is usually more complicated, there are in addition these important issues to consider:

- Shape parameterization
- Solving PDE
- Numerical issues

It is well-known that computing the gradient is necessary for a gradient-based shape optimization scheme. In the area of CFD, to compute the gradient is

not easy because it includes solving a PDE (partial differential equation) in most cases and most PDEs have no analytical solution. Therefore it is almost impossible to obtain the analytical formula of the gradient vector, because computing the gradient vector requires solving the governing PDE. Numerical methods for computing the gradient in shape optimization problems are listed below:

- Finite difference method
- The complex variable method
- Adjoint method
- Sensitivity analysis
- Automatic differentiation

Amongst these, the finite difference method is the most straightforward, but the computational cost is very expensive compared with the others because it requires $n + 1$ function evaluations in each optimization iteration (n represents the number of the design variables in optimization problems). The complex variable method is similar to the the finite difference method, see equation 2.41. The computational cost of the adjoint method is independent of the number of design variables but dependent on the number of the constraint equations. Hence this method is quite useful in cases which have a lot of design variables while having relatively few constraints. On the contrary, the computational cost of the sensitivity analysis is independent of the number of the constraint equations but dependent on the number of design variables. Therefore this method can be quite useful in cases which have

relatively few design variables. Given a shape optimization problem which has n design variables and m constraints, using the adjoint method requires solving m adjoint equations whilst using sensitivity analysis requires solving n equations.

Automatic differentiation is a method to numerically compute the derivative of a function specified by a computer program. It builds a new, augmented, program that evaluates the analytical derivatives along with the original computer program based on the principle that a given function can be differentiated by differentiating each line of the program. It has two modes: direct mode and reverse mode. These two modes are distinguished by how the chain rule is used to obtain derivatives through the computation. Reverse mode is particularly useful because the computational cost in this case is independent of the number of design variables.

Computing the gradient is researched heavily while shape parameterization is not studied extensively. Shape parameterization is important because it significantly influences the performance of the optimization scheme, especially in the case that the value of objective function is sensitive to the shape change, but not as much as the method used to compute the gradient because the gradient evaluation method decides the number of function calls in each optimization iteration.

Three shape parameterization methods are often of use in shape design. They are

- CAD-based
- Mesh-based
- Based on a set of reference shapes

In CAD-based shape parameterization, Bézier, B-spline, and NURBS are usually used to parameterize the geometry. In mesh based parameterization, the geometry model is not required for the optimization loop and is also the easiest way to start solving the optimization problem. In reference shape based parameterization, the geometry is represented as a linear combination of the reference shapes. Indeed the CAD-based shape parameterization can be regarded as a kind of reference shape based parameterization in theory as B-splines, NURBS (Non uniform Rational B-splines) can be regarded as special shape functions.

The numerical solution of the PDE is a necessary information in the optimization loop. The accuracy of the solution mainly depends on the method used to transform the PDE into a discrete problem and the method of solving such a discrete problem. To achieve high accuracy mesh refinement can be used, however global refinement may cause inefficiency, see [51].

Iterative methods are mainly used to solve a large scale PDE. For solving a system of linear equations (a linear PDE will change to be a system of linear equations after discretization), the convergence rate depends on the coefficient matrix. Therefore a preconditioner can be used to improve the

efficiency of solving the system.

$$A\mathbf{x} = \mathbf{b} \rightarrow M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$$

For some computationally expensive problems, a parallel code may be necessary; for example, in the case of 3-dimensional computational aerodynamic problems.

In this thesis, a new shape optimization method is developed. This method is applied to several shape optimization problems. In the method, the mesh refinement function is added to decrease the noise of the gradient because the discretization error can be significantly decreased by decreasing the mesh size, i.e. increasing the number of nodes or elements in the mesh. Furthermore, a method to detect and handle self-intersecting shapes is presented. During the development it was found that mesh refinement is particularly important in the case where the exact gradient is hard to access. The results demonstrate the success of this method.

An overview of the thesis is provided below:

Chapter Two covers the theoretical background of the shape optimization area. It begins with a literature review and then goes on to discuss basic optimization theory, the different methods for evaluating the gradients in shape design, the advantages and disadvantages associated with these methods, the geometry representation techniques, and finally the PDE solution techniques

which involve mesh refinement.

In the basic shape optimization theory, both unconstrained optimization and constrained optimization methods are discussed. Since the constrained optimization method can be regarded as an extension of unconstrained optimization methods constrained optimization methods are not discussed in great detail.

Typical unconstrained optimization methods include two parts: the methodology of finding a search direction and the methodology to determine the step size along the search direction. The former of these distinguishes the differences among the optimization methods. Search direction techniques employ the line search method or trust region method. They are used to determine the step size along the search direction.

Among the unconstrained optimization methods, the BFGS method (which is a branch of the Quasi-Newton method, BFGS stands for Broyden-Fletcher-Goldfarb-Shanno,) is one of the best methods because of its super linear convergence. However the weakness of this method is the requirement for the matrix operation. In each optimization iteration, it is necessary to update the Hessian matrix. Therefore, this method is not the first choice for a large number of design variables in terms of the computational cost.

In the shape optimization field, methods to compute the sensitivity have been extensively studied. Among these methods, the reverse mode of au-

tomatic differentiation and the adjoint method are particularly useful because the computational costs of these two methods are independent of the number of design variables.

The geometry representation methodology is also presented in the chapter. The CAD-based technique, such as B-splines, NURBS, seems more promising for the future because the CAD technique is a standard for industry applications to represent complex geometry information.

The PDE solution technique is usually an iterative scheme. Mesh refinement is also investigated because this approach can improve the accuracy of the PDE solution. Global mesh refinement and local mesh refinement are also further discussed.

Chapter Three focuses on the developed new method, the super-reduced method. This method is applied to the shape optimization problem, therefore it cannot be regarded as a general optimization method. Unlike other methods in the shape optimization field the new method focuses on the optimization iterations rather than the techniques used to compute the gradient.

The core idea of the new method is to introduce the B-spline knot insertion idea into the optimization scheme. The knot insertion algorithm is an algorithm to insert a number of control-points to a B-spline while keeping the geometry fixed. Therefore the initial number of design variables can be reduced to a minimum, potentially decreasing the number of optimization

iterations. When the solution is close to the optimum solution, the knot insertion algorithm starts and produces a large number of design variables as specified by the user. In doing so, an optimum shape can be found more efficiently in terms of the CPU time and number of optimization iterations. Furthermore, this will partly release the limitation of the BFGS method because it limits most of the optimization iterations to a matrix with small dimension. For a shape optimization problem with n design variables, the new method can limit most of the matrix operations within the dimension m , where m denotes the initial number of design variables, specified by the user, n denotes the final number of design variables and m can be much smaller than n .

In addition to the use of knot insertion, the new method also includes a super-reduced implementation which is used to get rid of the equality constraints. Thus, a constrained optimization problem is converted into an unconstrained optimization problem and the constraints are kept unchanged all the time. It is well known that solving a constraint optimization iteration is much more difficult than solving an unconstrained optimization problem. Therefore this approach will also decrease the CPU time.

With mesh refinement most of the iterations are done on the coarse grid with relatively few on the refined grid. This is particularly important when the exact gradient is difficult to obtain because mesh refinement can improve the accuracy level of the gradient by decreasing the discretization error.

Chapter Four contains the numerical applications of the new method in 2-dimensions. The first application of the method is a Poisson equation problem. Results for different objective functions, different initial numbers of control points, different accuracy levels, and different initial shapes are given. Furthermore, the results produced by the new method are compared to the results produced by the BFGS method, considered to be one of the best methods in optimization. The final results show that the new method is more efficient than BFGS method in terms of CPU time, optimization iterations, and number of function calls.

The second application of the method is an airfoil design problem. The performance of the new method is compared with the performance of the EXTREM method developed by Jacob, see [63] in two design cases, RAE 2822 and NACA 0012. In both of these two cases, the new method is shown to be more efficient than the traditional method in terms of the CPU time and number of function calls.

In addition to these applications, the use of high order B-splines and the multi-objective optimization have also been investigated. In summary, a new method has been developed for shape optimization problems and validation has been carried out on several numerical examples. Using the new method, it is possible to improve the efficiency of current methods being used as long as B-splines are used to describe the geometry.

Chapter Five contains some numerical applications of the new method

in 3-dimensions. Here, the geometry representation technique and geometry issues are emphasized. The final results show the convergence of the new method.

Finally, **Chapter Six** gives a summary of contributions made, conclusions and possible future development.

2 Literature review

2.1 Introduction

As early as the 1970's, Pironneau [44][45] first proposed a minimum drag problem in Stokes flow [1] and addressed variational methods (adjoint method), initially with the Stokes equations [44] and later with the incompressible Navier-Stokes equation [45]. However, due to the absence of CFD tools at that time he only derived the necessary optimality condition for several problems, leaving the numerical schemes aside.

Jameson [31] applied the adjoint method for optimal shape design of an airfoil using Euler equations. Some improvements for the efficiency on the Euler equation have been carried out by Iollo [29] and Ta'asan [58]. Examples of shape optimization problems governed by a potential equation can be found in [35][57].

The most important issue in shape optimization within gradient-based methods is how to compute the reduced gradients. The traditional way is to use the finite difference approach. This method has been employed by Lee [36][37]. The finite difference method with complex variable [3][56][60] is a comparatively better method although it requires a certain amount of code conversion and nearly double the memory. But both of these methods are extremely computationally expensive. This disadvantage motivates other more efficient methods, such as sensitivity analysis and adjoint methods.

With the sensitivity analysis [10][12][54], the computational efficiency and accuracy are improved significantly. However the computational cost is still proportional to the number of design variables. Therefore, this method is ideal for problems with many constraints but few design variables.

In the adjoint method, the adjoint equations are to be solved for each constraint and the computational cost is independent of the number of design variables. Therefore the advantage is appropriate at solving problems with many design variables but relatively few constraints. The adjoint method can be formulated on two different levels: the continuous level and the discrete level. The discrete adjoint method is more accurate but less easy to convert than the continuous adjoint method. Applications of this method can be found in [2][13][22][23] [29][30][32][55][62].

Besides the adjoint method and sensitivity analysis, automatic differentiation has been used widely to calculate the gradient vector. There are many software packages devoted to automatic differentiation written on the Fortran platform [9][50], with a few also written using the C and C++ platform [19]. Automatic differentiation uses two ways to compute the gradient, forward mode and reverse mode. By using the reverse mode, the cost is almost the same for different numbers of design variables and in this sense it is similar to the adjoint method.

Because the gradient-based method can generally find a local optimum rather than a global one, a genetic algorithm can also be used. The basic genetic

algorithm represents the design variables using discrete binary strings. A real-number-encoding GA is used to improve the time efficiency, see Holst [27]. In order to decrease the huge computation time, a hybrid genetic algorithm can be employed, see [41]. It has been shown that hybrid methods have priority over pure genetic algorithms in certain applications in terms of computation time. A constrained genetic algorithm is employed in [59]. GA's weakness lies in its huge amount of object function calls. Hence a parallel GA idea is presented, see [61].

In this research subject the question of how best to represent the geometry is an open issue. Parameterization [16][52] is a widely accepted technique. For example, Burkardt, [12], uses one parameter to represent the geometry. An interesting approach for shape representation is to use the coordinates of boundary points to represent the geometry. This approach is simple to implement but causes a large design space and it cannot easily maintain smooth geometry.

The Bézier family is well suited for some shape optimization problems with a few number of design variables, see [5]. However, complex geometry requires a high degree Bézier form and in this case B-splines are a better choice. Using B-splines [18][47] makes it easy to extend the result to other CAD-based libraries. An application can be found in [2]. In this work the control points will be regarded as design variables.

All of these above shape representations are quite simple to implement in

the 2-dimensional case. However for some 3-dimensional problems, for example [49], the above shape representations may cause self-intersecting volumes, particularly when the problem is unbounded. A very good method called the *level set method*, originally presented by Osher and Sethian [43], can avoid this situation. This method does not try to parameterize the geometry but rather uses a level set of a continuous function. The desired shapes produced by level set methods are given by the zero level set. The geometry evolution is applied to the level set functions rather than the real initial geometry. However, this method is not as efficient as others when the geometry is relatively simple. More details can be found in [11], [42].

The computational cost for shape optimization is usually very expensive, especially when the governing equation is a non-linear partial differential equation or the required geometry is a 3-dimensional volume. The computational cost can be decreased by choosing different ways of evaluating the gradients and using different parameterizations. Furthermore some other methods may help to decrease the computational cost under certain circumstances. For example, the CPR algorithm, [15], can decrease the computational cost when the Jacobian matrix, J , has some zero elements or known values. An application is given in [14].

There are still some other important issues in shape optimization, such as mesh refinement, the solution of non-linear PDE's, the inexact gradient method etc. For more on this, readers are referred to [40].

2.2 Optimization theory

The basic optimization theory presented here can be found in most text books on optimization, for example [6], [19]. Optimization methods in practice can be divided into two categories [62]: non-gradient methods and gradient methods. In the former there is no gradient information required, for example the basic genetic algorithm (GA). In the latter type, the gradient information of the objective function with respect to the design variables is of great importance. The ideal way to obtain the gradient information is to differentiate the analytical solution. However it is almost impossible to find the exact solution in most practical cases, such as in the CFD domain, where PDEs are involved.

The gradient based method has the advantage of efficiency in that it can find the local optimum with a finite number of iterations and it limits to find only a local optimum, not necessarily the global optimum. On the other hand the non-gradient method (Global method) can find the global optimum although the computation cost is usually expensive. The reason is that the global method needs to evaluate the objective function a huge number of times. There is a theoretical result which links the global optimum and the local optimum: *if the defined region R is a convex region and the objective function $F(\mathbf{x})$ is a convex function in R , then any local minimiser $\mathbf{x}^*$ is also a global minimiser.* However this result contributes little to optimization knowledge since the pre-conditions are too strong to be true in many applications.

Optimization problems can be divided into two different cases:

- Unconstrained optimization
- Constrained optimization

For each of these, there are some corresponding methods to be used. In general, the method for solving constrained optimization problems can be regarded as an extension of the methods used for solving unconstrained optimization problems.

2.2.1 Unconstrained optimization

The basic idea of different unconstrained optimization schemes is the same; that is first finding a descent direction and then determining a proper step size.

Steepest descent method

The method of steepest descent is the simplest one among gradient methods. The choice of descent direction is where F , the objective function, decreases most quickly, which is obviously in the direction opposite to the gradient ∇F at $\mathbf{x}_i$. The search starts at an arbitrary point $\mathbf{x}_0$ and then goes down along the search direction until the point is close enough to the optimum. In

other words, the iterative procedure is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}(\mathbf{x}_k), \quad (2.1)$$

where $\mathbf{g}(\mathbf{x}_k)$ is the gradient at one given point, α_k is the step size along the descent direction (search direction), $\mathbf{x}_k, \mathbf{x}_{k+1}$ refers the point at k -th, $k+1$ -th optimization iteration respectively.

One question remaining now is how big should the step taken in that direction be; i.e. what is the value of α_k ? Obviously the purpose is to move to the point where the objective function F takes on a minimum value, which is where the directional derivative [31] (the directional derivative of $\nabla F(\mathbf{x}_{k+1})$ along direction $-\mathbf{g}(\mathbf{x}_k)$ which needs to be normalized) is zero. The directional derivative is given by

$$-\nabla F(\mathbf{x}_{k+1})^T \cdot \frac{\mathbf{g}(\mathbf{x}_k)}{\|\mathbf{g}(\mathbf{x}_k)\|} = \frac{d}{d\alpha_k} F(\mathbf{x}_{k+1}) = 0 \quad (2.2)$$

In order to solve this problem, the Taylor series is used as an approximation:

$$\begin{aligned} F(\mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)) &\approx F(\mathbf{x}_k) - \alpha_k [\nabla F(\mathbf{x}_k)]^T \nabla F(\mathbf{x}_k) \\ &+ \frac{1}{2} \alpha_k^2 [\nabla F(\mathbf{x}_k)]^T H(\mathbf{x}_k) \nabla F(\mathbf{x}_k) \end{aligned} \quad (2.3)$$

We differentiate the above equation with respect to α_k , giving the equation below:

$$\alpha_k = \frac{[\nabla f(x_k)]^T \nabla f(x_k)}{[\nabla f(x_k)]^T H(x_k) \nabla f(x_k)}. \quad (2.4)$$

The method used to determine the value of the step size above is the so-called exact line search method for a quadratic model, not for a general nonlinear function.

Quasi-Newton's method

The most popular method in unconstrained optimization is the BFGS method, which is a branch of the Quasi-Newton method. The central idea underlying the Quasi-Newton method is to use an approximation of the inverse Hessian. Suppose two points $\mathbf{x}_k, \mathbf{x}_{k+1}$ are given, then the gradient vectors at point $\mathbf{x}_k$, and point $\mathbf{x}_{k+1}$ are defined, $\mathbf{g}(\mathbf{x}_k), \mathbf{g}(\mathbf{x}_{k+1})$, respectively. Further let $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{q}_k = \mathbf{g}(\mathbf{x}_{k+1}) - \mathbf{g}(\mathbf{x}_k)$. Now use Taylor's series for $\nabla F(\mathbf{x}_k)$ (or $\mathbf{g}(\mathbf{x}_k)$) then

$$\mathbf{q}_k = \mathbf{g}(\mathbf{x}_{k+1}) - \mathbf{g}(\mathbf{x}_k) = H(\mathbf{x}_k) \mathbf{p}_k + \mathbf{O}(\|\mathbf{p}_k\|)$$

In order to correctly relate the difference, H_{k+1} is chosen to satisfy the following condition.

$$H_{k+1}^{-1} \mathbf{q}_k = \mathbf{p}_k \tag{2.5}$$

This is called the Quasi-Newton condition.

There are two formulae used very widely in unconstrained optimization:

1. Davidon-Fletcher-Powell (DFP) formula

2. Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula

The DFP update formula is given by the equation below:

$$B_{k+1} = B_k + \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k} - \frac{B_k \mathbf{q}_k \mathbf{q}_k^T B_k}{\mathbf{q}_k^T B_k \mathbf{q}_k} \quad (2.6)$$

where B represents the inverse of the Hessian matrix. The BFGS update formula is obtained from the DFP update formula by interchanging B_k with H_k and interchanging $\mathbf{q}$ with $\mathbf{p}$ at the same time. Therefore:

$$H_{k+1} = H_k + \frac{\mathbf{q}_k \mathbf{q}_k^T}{\mathbf{q}_k^T \mathbf{p}_k} - \frac{H_k \mathbf{p}_k \mathbf{p}_k^T H_k}{\mathbf{p}_k^T H_k \mathbf{p}_k} \quad (2.7)$$

We use the equation below to avoid computing the inverse matrix:

$$B_{k+1} = B_k + \frac{1 + \mathbf{q}_k^T B_k \mathbf{q}_k}{\mathbf{q}_k^T \mathbf{p}_k} \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k} - \frac{\mathbf{p}_k \mathbf{q}_k^T B_k + B_k \mathbf{q}_k \mathbf{p}_k^T}{\mathbf{q}_k^T \mathbf{p}_k}. \quad (2.8)$$

Numerical experiments have shown that the BFGS formula has superior performance over the DFP formula, and hence, BFGS is often preferred over DFP. These two formulae lead to a whole collection of updates known as the Broyden family which are defined by:

$$B^\phi = (1 - \phi) B^{\text{DFP}} + \phi B^{\text{BFGS}} \quad (2.9)$$

where ϕ is a parameter that may take any real value.

Conjugate gradient method

Besides the Quasi-Newton method, some other methods are still used. For example, the *conjugate gradient method*. The search direction $\mathbf{s}_i$ in this method is defined below:

$$\mathbf{s}_1 = -\mathbf{g}_1$$

$$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{s}_k$$

where $\mathbf{g}$ is the gradient vector, and

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k}.$$

Conjugate gradient methods are applicable to large scale problems because no matrix calculations are required. The aim of using the conjugate gradient method is to achieve both efficiency and reliability by associating conjugacy properties with the steepest descent method. It uses conjugate directions instead of the local gradient for going downhill.

As with other methods, the new point is generated by the previous point in each iteration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k,$$

where $\mathbf{s}_k$ is the search direction, and α is the step size which can be determined by a line search algorithm. This is the basis of the conjugate gradient method. Compared with the Quasi-Newton method, the conjugate gradient method is both less efficient and less robust. However there is one particular

advantage of this method, namely the particularly simple form of the equation which requires no matrix operations to form the search direction. Thus conjugate gradient methods are applicable for large problems and work very well. There are some modified conjugate methods, but these may cause the main advantage (no matrix calculations required) to disappear.

The content of the above only discusses how to select the search direction. A line search algorithm is needed in order to determine how large the amount of the step size should be.

2.2.2 Line search algorithm and trust region method

A line search algorithm, or the trust region method, is an important issue in optimization schemes because it considerably influences the performance of the optimization methods. Based on [20] there is no conclusion about whether the trust region method is superior to the line search method. In the case where the gradient information is not available, then there is little theory on how to determine the step size in a search direction.

Line search methods

There are two approaches to line search methods: one is the sectioning or bracketing phase, the other is interpolation. The sectioning phase finds an interval which contains an interval of acceptable points for the optimization

iteration. Interpolation uses a polynomial function to fit the desired function with known values, and then find the step size which can minimize the polynomial function.

The simplest line search algorithm is the *bisection method*.

- initialize step size α_k , say 1.0
- do, $\alpha_k = \frac{\alpha_k}{2}$, until $F(\mathbf{x}_k + \alpha_k \mathbf{s}_k) < F(\mathbf{x}_k)$

An exact line search algorithm has already been mentioned before, which requires information from the Hessian matrix. The utility of the exact line search algorithm is because that it can achieve global convergence when combined with the steepest descent method. However, using a exact line search algorithm is so expensive to carry out. Furthermore, some methods' convergence rates do not require the exact line search, such as in the Quasi-Newton method.

The line search method aims to find a step size which makes the objective function decrease. However this is not enough for two reasons. First, the value of the objective function may decrease too slowly compared with the step size, i.e, it is not a sufficient decrease. Second the step size may be too small compared with the initial rate of the decrease of the objective function.

Two conditions on the step size can solve the above problems, i.e.

$$f(\theta) \leq f(0) + \theta \rho f'(0) \quad (2.10)$$

$$f(\theta) \geq f(0) + \theta(1 - \rho)f'(0) \quad (2.11)$$

where ρ is in $(0, 0.5)$, and f represents a function of the step size. This is the so called Goldstein condition. In the above equation array, $\rho < 0.5$ allows the property that the minimum value of a quadratic function is acceptable. But when $f(\theta)$ is not a quadratic function the second condition may exclude the minimum point of $f(\theta)$.

Therefore the second condition can be replaced by the following equation:

$$\mathbf{g}_{k+1}^T \mathbf{q}_k \geq \rho_2 \mathbf{g}_k^T \mathbf{q}_k$$

where $\mathbf{q}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$.

This gives the Wolfe-Powell's condition:

$$f(\theta) \leq f(0) + \theta \rho_1 f'(0) \quad (2.12)$$

$$\mathbf{g}_{k+1}^T \mathbf{q}_k \geq \rho_2 \mathbf{g}_k^T \mathbf{q}_k. \quad (2.13)$$

It has been proved in [20] that a line search method based on Goldstein's condition or Wolfe-Powell's condition will achieve global convergence.

A mixed interpolation line search method

Below, we study a mixed interpolation line search method which is often used in optimization algorithms. In the line search method, the step size is usually initialized as 1.0 in the search direction because this value will lead to the quadratic convergence when the estimated solution is sufficiently close to the analytical solution. If this fails, the value $f(1)$ is computed. Therefore the quadratic model is constructed as follows:

$$f(\theta) = [f(1) - f(0) - f'(0)]\theta^2 + f'(0)\theta + f(0)$$

hence, f has the minimum value when

$$\theta = \frac{0.5 \times f'(0)}{f(1) - f(0) - f'(0)}$$

If a cubical interpolation is used, then the function can be constructed as below:

$$f(\theta) = a\theta^3 + b\theta^2 + f'(0)\theta + f(0)$$

To find the value of the coefficients a , b , values of $f(\theta_1)$ and $f(\theta_2)$ are needed:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{\theta_2 - \theta_1} \begin{pmatrix} \frac{1}{\theta_1^2} & \frac{-1}{\theta_2^2} \\ \frac{-\theta_2}{\theta_1^2} & \frac{\theta_1}{\theta_2^2} \end{pmatrix} \begin{pmatrix} f(\theta_1) - f(0) - f'(0)\theta_1 \\ f(\theta_2) - f(0) - f'(0)\theta_2 \end{pmatrix} \quad (2.14)$$

Therefore, the minimum of function f is at

$$\theta = \frac{-b + \sqrt{b^2 - 3af'(0)}}{3a}$$

Trust region method

In the above, the Hessian matrix or the approximated Hessian matrix is required to be positive definite. To circumvent the difficulty of a non-positive definite Hessian matrix, the trust region method or the restricted step method is motivated.

The idea of the trust region method is to minimize the quadratic model subject to an explicit bound on the step size, that is:

$$\min m(\mathbf{x}_c + h) = F(\mathbf{x}_c) + \nabla F(\mathbf{x}_c)^T h + \frac{1}{2} h^T B h \quad \text{subject to } \|h\| \leq r \quad (2.15)$$

where $\mathbf{x}_c$ refers to the current point, and h denotes the step size vector, not the step size. The solution for the above constrained optimization problem is then given by

$$h(\eta) = -(B + \eta I)^{-1} \nabla F(\mathbf{x}_c) \quad (2.16)$$

where η is chosen such that $\|h\| = r$ if $h(0) > r$.

A question that arises is: how good is the approximation? Written

$$\Delta F^k = F^k - F(\mathbf{x}_k + h),$$

$$\Delta m^k = F^k - m(\mathbf{x}_k + h);$$

then the ratio $\frac{\Delta F^k}{\Delta m^k}$ is a standard measure of how good the approximation is.

The closer the ratio to 1.0, the better the approximation.

2.2.3 Constrained optimization

In practice constrained optimization is much more important than unconstrained optimization. However the method of solving constrained optimization problems can be regarded as an extension of the method solving unconstrained optimization problems.

The most important concept in constrained optimization theory is that of Lagrange multipliers. For a general equality constrained optimization problem (with m equality constraints),

$$\min F(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n, \quad \text{subject to} \quad \mathbf{c}_i(\mathbf{x}) = 0 \quad (2.17)$$

the necessary condition is:

$$\begin{aligned} \mathbf{c}_i(\mathbf{x}^*) &= \mathbf{0} \\ \mathbf{g}(\mathbf{x}^*) &= J(\mathbf{x}^*)^T \boldsymbol{\lambda} \\ \mathbf{v}^T W(\mathbf{x}^*, \boldsymbol{\lambda}^*) \mathbf{v} &\geq 0 \quad \forall \mathbf{v} \in \mathbb{R}^{n+m} \end{aligned} \quad (2.18)$$

where J is the Jacobian matrix of the constraint $\mathbf{c}_i$ with respect to design variables $x_1, x_2, \dots, x_n$,

$$J = \begin{pmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_1}{\partial x_2} & \cdots & \frac{\partial c_1}{\partial x_n} \\ \frac{\partial c_2}{\partial x_1} & \frac{\partial c_2}{\partial x_2} & \cdots & \frac{\partial c_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial c_m}{\partial x_1} & \frac{\partial c_m}{\partial x_2} & \cdots & \frac{\partial c_m}{\partial x_n} \end{pmatrix}$$

W is the Hessian matrix of the Lagrangian function:

$$L = F(\mathbf{x}) - \lambda^T \mathbf{c}(\mathbf{x}) \quad (2.19)$$

and λ is the so-called Lagrange multiplier. By using a Lagrangian function, the first-order necessary condition becomes:

$$\nabla L = \begin{pmatrix} \nabla_{\mathbf{x}} L \\ \nabla_{\lambda} L \end{pmatrix} = \mathbf{0} \quad (2.20)$$

The above equation is very useful. Two Lagrangian functions are used widely:

- Augmented Lagrangian function:

$$L_A = F(\mathbf{x}) - \lambda^T \mathbf{c}(\mathbf{x}) + \frac{\rho}{2} \mathbf{c}(\mathbf{x})^T \mathbf{c}(\mathbf{x}), \quad (2.21)$$

where the penalty parameter ρ is positive.

- Projected Lagrangian function:

$$L_c = F(\mathbf{x}) - \lambda^T \mathbf{c}(\mathbf{x}) + \lambda^T J \mathbf{x}, \quad (2.22)$$

where J is the Jacobian matrix. Generally it is assumed $\det(J) \neq 0$.

In the augmented Lagrangian function, the penalty coefficient is used to prevent violating the constraints. Therefore the constrained optimization problem changes to be an unconstrained optimization problem of minimizing the augmented Lagrangian function.

Another choice of solving the constrained optimization is using by SQP (sequential quadratic programming):

$$\min m(\mathbf{x}_c + h) = f(\mathbf{x}_c) + \nabla f(\mathbf{x}_c)^T h + \frac{1}{2} h^T H h \quad \text{subject to } Jh = -c \leq r \quad (2.23)$$

where H is the Hessian matrix and J is the Jacobian matrix.

The QP method

The material of this section is based on the following reference: [20]. Here the purpose is to find the solution of the problem:

$$\min_{\mathbf{x}} \mathbf{g}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T H \mathbf{x} \quad \text{subject to } A\mathbf{x} = \mathbf{b} \quad (2.24)$$

It is assumed that H is positive definite symmetric matrix and A is a full rank matrix. There is thus only one solution and the constraints are equality constraints.

Let Y and Z be $n \times m$ and $n \times (n - m)$ matrix respectively such that

$[Y : Z]$, a $n \times n$ combining matrix by moving each column of matrix Z into matrix Y in a sequential order, is non-singular, and in addition let $AY = I$ and $AZ = 0$. The solution of the equation $Ax = b$ is given by:

$$\mathbf{x} = Y\mathbf{b} + Z\mathbf{y} \quad (2.25)$$

where $\mathbf{y}$ can be any vector. Therefore, the following equation is addressed:

$$A\mathbf{x} = A(Y\mathbf{b} + Z\mathbf{y}) = AY\mathbf{b} + AZ\mathbf{y} = \mathbf{b}.$$

The matrix Z has $n - m$ linearly independent columns $z_1, \dots, z_{n-m}$ which are inside the null-space of A and therefore act as basis vectors (or reduced coordinate directions) for the null space. At any point $\mathbf{x}$ any feasible correction δ can be written as:

$$\delta = Z\mathbf{y} = \sum_{i=1}^{n-m} z_i y_i \quad (2.26)$$

where $y_1, \dots, y_{n-m}$ are the components (or reduced variables) in each reduced coordinate direction (see the following figure). Combining equations 2.24 and 2.25, The following reduced quadratic function is obtained:

$$\psi(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T Z^T H Z \mathbf{y} + (\mathbf{g} + H Y \mathbf{b})^T Z \mathbf{y} + \frac{1}{2}(\mathbf{g} + H Y \mathbf{b})^T Y \mathbf{b} \quad (2.27)$$

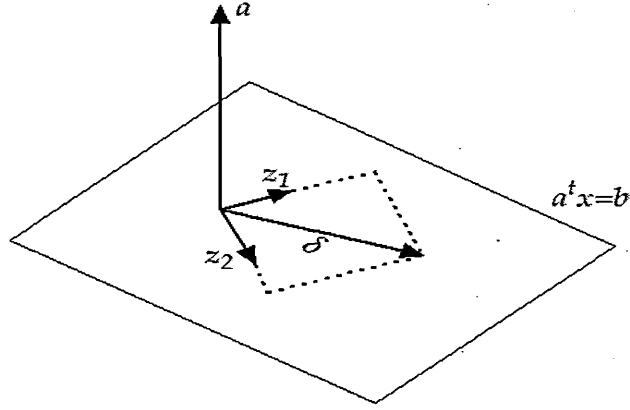


Figure 1: The null-space of A

If $Z^T H Z$ is positive definite then a minimizer $\mathbf{y}^*$ for equation 2.27 exists which solves the linear system:

$$(Z^T H Z)\mathbf{y} = -Z^T(\mathbf{g} + H\mathbf{Y}\mathbf{b}) \quad (2.28)$$

The solution $\mathbf{y}^*$ is computed using Cholesky factorization. Once $\mathbf{y}^*$ is known $\mathbf{x}^*$ can be computed by using equation 2.25, and $\mathbf{g}^*$ using the equation $\mathbf{g}^* = H\mathbf{x}^* + \mathbf{g}$. Now recall the equation

$$\mathbf{g}(\mathbf{x}) = \sum_{j \in E} \lambda_j \nabla \mathbf{c}_j(\mathbf{x}) \quad (2.29)$$

and compute λ^* : From the above equation 2.29, the following equation is obtained:

$$\mathbf{g}^* = A^T \lambda^*. \quad (2.30)$$

If we pre-Multiply 2.30 by Y^T and, using $Y^T A^T = I$, the following equation is addressed:

$$Y^T \mathbf{g}^* = \lambda^*. \quad (2.31)$$

Depending on the choice of Y and Z , a number of methods exist. Y and Z can be obtained by a QR factorization of the matrix A^T . This can be written:

$$A^T = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_1 \quad Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_1 R \quad (2.32)$$

where $Q \in \mathbb{R}^{n \times n}$ is an orthogonal matrix, $R \in \mathbb{R}^{m \times m}$ is an upper triangular matrix, $Q_1 \in \mathbb{R}^{n \times m}$ and $Q_2 \in \mathbb{R}^{n \times (n-m)}$. The choices

$$Y = Q_1(R^t)^{-1}, \quad Z = Q_2, \quad (2.33)$$

have the correct properties. Moreover the vector Yb in equation 2.25 is orthogonal to the constraints. The reduced coordinate directions $\mathbf{z}_i$ are also mutually orthogonal. Yb is calculated by forward substitution in $R^t \mathbf{u} = \mathbf{b}$ followed by $Yb = Q_1 \mathbf{u}$. The multipliers λ^* are calculated by backward substitution in $R\lambda^* = Q_1^t \mathbf{g}^*$.

The SQP method

The material here is based on the following references: [20]. The SQP algorithm is:

Algorithm: the SQP algorithm

1. Set $k := 1$
2. Solve the QP subproblem to determine δ_k
3. Let λ_{k+1} be the vector of the Lagrange multiplier of the
linear constraints obtained from the QP
4. Compute the step size α_k and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \delta_k$
5. Compute H_{k+1} from H_k using a Quasi-Newton formula
6. Increment k . Stop if $\nabla L(\mathbf{x}_k, \lambda_k) \approx 0$. Otherwise, go to step 2.

A detailed discussion of each step is given as follows. The QP sub problem has already been described earlier in this section.

The method to compute the length of the step along the direction δ_k is list as follows: From the QP, a direction δ_k is obtained. To have a more robust code, the first Wolf condition is applied which is recalled here:

$$f(\alpha) \leq f(0) + \rho \alpha f'(0), \quad \rho \in (0, \frac{1}{2}), \quad (2.34)$$

where $f(\alpha) = f(\mathbf{x}_k + \alpha \delta_k)$. This condition ensures a “sufficient enough” reduction of the objective function at each step. Unfortunately, in the constrained case, sufficient reduction of the objective function is not enough.

The reduction of the infeasibility must also be ensured. Therefore, a modified form of the first Wolf condition is used where $f(\alpha) = \phi(\mathbf{x}_k + \alpha\delta_k)$ and $\phi(\mathbf{x})$ is a merit function. In the optimizer the l_1 merit function is used:

$$\phi(\mathbf{x}) = f(\mathbf{x}) + \frac{1}{\mu} \|c(\mathbf{x})\|_1 \quad (2.35)$$

where $\mu > 0$ is called the penalty parameter. A suitable value for the penalty parameter is obtained by choosing a constant $K > 0$ and defining μ at every iteration to be

$$\frac{1}{\mu} = \|\lambda_{k+1}\|_\infty + K \quad (2.36)$$

In equation 2.34, it is necessary to calculate $f'(0)$: the directional derivative of ϕ in the direction δ_k at x_k :

$$f'(0) = D_{\delta_k} \phi(\mathbf{x}_k) = \delta_k^t \mathbf{g}_k - \frac{1}{\mu} \left(\delta_k^t a_i + \sum_{i \in \mathcal{Z}_k} \max(-\delta_k^t a_i, 0) \right), \quad (2.37)$$

where $\mathcal{F}_k = \{i : c_i(x_k) < 0\}$, $\mathcal{Z}_k = \{i : c_i(x_k) = 0\}$, $g_k = \nabla f(x_k)$, $a_i = \nabla c_i(x_k)$. The algorithm which computes the length of the step is thus:

Algorithm: a sample line search algorithm used in SQP

1, Set $\alpha_k := 1$, $x_k :=$ current point, $\delta_k :=$ direction of search

2, Test the Wolf condition equation 2.34:

$$\phi(x_k + \alpha_k \delta_k) \leq \phi(x_k) + \rho \alpha_k D_{\delta_k} \phi(x_k)$$

True: Set $x_{k+1} := x_k + \alpha_k \delta_k$ and go to 3.

False: Set $\alpha_k = 0.7 \times \alpha_k$ and return to the beginning of 2.

3, We have successfully computed the length $\|\alpha_k \delta_k\|$ of the step.

Penalty functions

For a general non-linear optimization problem, (which means both the objective and constraints functions may be non-linear,) the penalty function is first used. However this method suffers from computational disadvantages and is not so efficient. The basic penalty function for the equality constraints problem is:

$$\phi(\mathbf{x}, \theta) = F(\mathbf{x}) + \frac{1}{2} \theta \mathbf{c}(\mathbf{x})^T \mathbf{c}(\mathbf{x}). \quad (2.38)$$

The aim is to find a local minimizer $\mathbf{x}^*$ that minimises $\phi(\mathbf{x}, \theta)$ as $\theta \rightarrow \infty$.

The general implementation is stated below:

1. Choose a fixed sequence $\{\theta^{(k)}\} \rightarrow \infty$, typically $1, 10, 10^2, 10^3, 10^4, \dots$

2. For each $\theta^{(k)}$ find a local minimizer, $\mathbf{x}(\theta^{(k)})$ to $\min_x \phi(\mathbf{x}, \theta^{(k)})$
3. Terminate when $c(\mathbf{x}(\theta^{(k)}))$ is sufficient small, i.e. meets the termination condition.

2.3 Computing gradient in shape optimization

From a mathematical point of view, the shape optimization problem in CFD [1][19][38] (Computational Fluid Dynamics) could be regarded as a PDE-constrained optimization problem because the governing equations in CFD are PDEs. (Partial Differential Equations) The shape design problem can be expressed as:

$$\min_{\alpha} F(\alpha, \mathbf{U}(\alpha)), \quad \text{subject to} \quad \mathbf{L}(\alpha, \mathbf{U}(\alpha)) = 0 \quad (2.39)$$

Where $\mathbf{U} = (U_1, U_2, \dots, U_m)$ called *state variables*, $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ called *design variables*, $\mathbf{L} = (L_1, L_2, \dots, L_m)$ are the constraints.

For the gradient-based method, the gradients of the objective function with respect to design variables represent the key information. Obviously the ideal approach is to try to obtain the analytical solution first and then differentiate, but for most PDEs it is impossible to obtain an analytical solution, even for the simple PDEs such as Poisson's equation with complex geometry [17]. Some methods to compute the gradient information are presented below.

2.3.1 Finite difference approach

The classical and most straightforward method to compute the gradients is the *finite difference approach*. This method is widely acceptable for a black-box code [57] where the user does not have access to the PDE source:

$$\frac{dF}{d\alpha_i} \approx \frac{F(\alpha^i + \epsilon, \mathbf{U}(\alpha^i + \epsilon)) - F(\alpha^i, \mathbf{U}(\alpha^i))}{\epsilon} \quad (2.40)$$

where $\alpha^i + \epsilon$ represents $(\alpha_1, \alpha_2, \dots, \alpha_i + \epsilon, \dots, \alpha_n)$, α^i represents $(\alpha_1, \alpha_2, \dots, \alpha_n)$.

Three well-known difficulties with this method are listed below:

- The proper value of ϵ
- The round-off error caused by the subtraction of two nearly equal terms
- The huge computational cost, proportional to the size of the state space times the design space

In order to avoid the choice of ϵ and also to avoid the subtraction error, a *complex variable method* ([3][56][60]) can be used:

$$\frac{dF}{d\alpha_i} = \frac{\text{Im}F(\alpha_i + \epsilon, \mathbf{U}(\alpha_i + \epsilon))}{\epsilon} + O(\epsilon) \quad (2.41)$$

where $\text{Im}F$ represents the imaginary part of F .

Note: the third difficulty as mentioned above still exists.

2.3.2 Sensitivity analysis

Consider the general optimization problem as stated above. The gradients of the state variables with respect to the design variables, (called the *sensi-*

tivity), can be obtained by numerically solving the following:

$$\left[\frac{\partial \mathbf{L}}{\partial \alpha} \right] + \left[\frac{\partial \mathbf{L}}{\partial \mathbf{U}} \right] \left[\frac{\partial \mathbf{U}}{\partial \alpha} \right] = \mathbf{0}, \quad (2.42)$$

to give:

$$\left[\frac{\partial \mathbf{U}}{\partial \alpha} \right] = - \left[\frac{\partial \mathbf{L}}{\partial \mathbf{U}} \right]^{-1} \left[\frac{\partial \mathbf{L}}{\partial \alpha} \right].$$

Therefore, the gradients of the objective function with respect to the design variables are given by:

$$\frac{dF}{d\alpha} = - \left(\left[\frac{\partial \mathbf{L}}{\partial \mathbf{U}} \right]^{-1} \left[\frac{\partial \mathbf{L}}{\partial \alpha} \right] \right)^T \frac{\partial F}{\partial \mathbf{U}} + \frac{\partial F}{\partial \alpha} \quad (2.43)$$

$\frac{dF}{d\alpha}$ is also called the *reduced gradient* (notation: $\frac{\partial \hat{F}}{\partial \alpha}$). $\frac{\partial F}{\partial \mathbf{U}}$ and $\frac{\partial F}{\partial \alpha}$ could be computed by finite differences.

Note: No information about the objective function is included in the computation of sensitivity.

2.3.3 Adjoint method

Consider a variation in the objective function F which is caused by a small variation in the design variables. ($\alpha \rightarrow \alpha + \epsilon \tilde{\alpha}$)

$$\delta F = F(\alpha + \epsilon \tilde{\alpha}, \mathbf{U}(\alpha + \epsilon \tilde{\alpha})) - F(\alpha, \mathbf{U}(\alpha)) = \epsilon (\tilde{\alpha}^T F_{\alpha} + \tilde{\mathbf{U}}^T F_{\mathbf{U}}) + O(\epsilon^2) \quad (2.44)$$

where $\tilde{\mathbf{U}}$ and $\tilde{\alpha}$ are related by:

$$\tilde{\mathbf{U}}^T \mathbf{L}_{\mathbf{U}}^T + \tilde{\alpha}^T \mathbf{L}_{\alpha}^T = 0,$$

Therefore an arbitrary vector $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m)^T$ can be chosen

$$(\tilde{U}^T \mathbf{L}_U^T + \tilde{\alpha}^T \mathbf{L}_\alpha^T) \lambda = 0. \quad (2.45)$$

After several steps (similar example of this can be found in Appendix F.) we obtain the equation below:

$$\delta F = \epsilon \tilde{\alpha}^T (F_\alpha + \mathbf{L}_\alpha^T \lambda) + \epsilon \tilde{U}^T (F_U + \mathbf{L}_U^T \lambda) + O(\epsilon^2), \quad (2.46)$$

which is true for all λ . Now a proper choice for λ is given by the *Adjoint Equation*

$$\frac{\partial F}{\partial \mathbf{U}} + \left[\frac{\partial \mathbf{L}}{\partial \mathbf{U}} \right]^T \lambda = \mathbf{0} \quad (2.47)$$

The gradient information $\frac{dF}{d\alpha}$ is therefore:

$$\frac{dF}{d\alpha} = \left[\frac{\partial \mathbf{L}}{\partial \alpha} \right]^T \lambda + \frac{\partial F}{\partial \alpha} \quad (2.48)$$

At a minimum, the optimality condition (necessary condition) is:

$$\begin{aligned} \mathbf{L}(\mathbf{U}(\alpha), \alpha) &= \mathbf{0}, \\ \mathbf{L}_U^T \lambda + F_U &= \mathbf{0}, \\ \mathbf{L}_\alpha^T \lambda + F_\alpha &= \mathbf{0}. \end{aligned} \quad (2.49)$$

All of the above equations could be deduced another way by constructing a Lagrangian function. Of course, the mathematical expression of reduced

gradients is exactly the same after rearranging:

$$\frac{dF}{d\alpha} = - \left(\begin{bmatrix} \frac{\partial \mathbf{L}}{\partial \mathbf{U}} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial \mathbf{L}}{\partial \alpha} \end{bmatrix} \right)^T \frac{\partial F}{\partial \mathbf{U}} + \frac{\partial F}{\partial \alpha}$$

Note: There is no information about the design variables involved in the adjoint equation. This observation indicates that the computational cost of solving the adjoint equation is independent of the number of design variables. Therefore this method is particularly efficient for a large-scale design variable problem, although the cost is still proportional to the number of constraints.

2.3.4 Automatic differentiation

Automatic differentiation (AD) is a tool to numerically compute the derivative of a function specified by code. There are two traditional methods to do the differentiation

- symbolically differentiate the function as an expression, and then evaluate it at the required point
- use numerical differentiation

However, using symbolic differentiation is low speed and also it is difficult to convert the code into a single mathematical expression. Furthermore, if higher order derivatives are required, then the final results would be very complicated and time consuming.

For using numerical differentiation, such as the finite differences, the difficulties are round-off error and cancellation. Furthermore, it shares the

same problem with the symbolic differentiation. To obtain the higher order derivatives will bring the complexity.

Automatic differentiation can solve all of the above problems. In the automatic differentiation(AD) method, it builds a new, augmented, program that computes the analytical derivatives information of the function with respect to the variables, along with the original program. This new program can be called as the differentiated program. Precisely, each time the original program holds some value t , the differentiated program holds an additional value dt , the differential of t . Moreover, each time the original program performs some operation, the differentiated program performs additional operations dealing with the differential values.

AD's principle is based on a fact, that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations such as additions, times or elementary functions such as log. Therefore, the whole computer program can be differentiated with the help of the chain rule. (see also [39][40])

By applying the chain rule of derivative calculus repeatedly to the operations in the original code which are used to compute the desired function, derivatives of arbitrary order can be computed automatically. This process yields exact (to numerical accuracy) derivatives.

There are two approaches developed in automatic differentiation: the for-

ward mode and the reverse mode. These two modes are distinguished by how the chain rule is used to propagate derivatives through the computation. The forward mode propagates derivatives of intermediate variables with respect to the independent variables. The reverse mode traverses the chain rule from left to right. The forward mode is quite straightforward. The reverse mode usually needs more memory.

In the reverse mode of automatic differentiation, the computational cost is independent of the number of design variables.

To implement AD, there are two choice:

- overloading

This is a C++ style. Each real number is replaced by a pair of real numbers, the second holding the differential. Each elementary operation on real numbers is overloaded, i.e. internally replaced by a new one, working on pairs of reals, that computes the value and its differential. The advantage is that the original program is virtually unchanged, since everything is done at compile time. The drawback is that the resulting program runs slowly because it constantly builds and destroys pairs of real numbers. Moreover, it is very hard to implement the "reverse mode" with overloading.

- source transformation

consists in adding into the program the new variables, arrays, and data structures that will hold the derivatives, and in adding the new

instructions that compute these derivatives. The advantage is that the resulting program can be compiled into an efficient code, and the "reverse mode" is possible. The drawback is that this is an enormous transformation, that cannot be done by hand on large programs.

Automatic differentiation tools are mainly developed on the Fortran platform such as ADIFOR [9], Odyssée [50] etc. With some software on C and C++ platform, ADOL-C [21]. Some software can even deal with the non-differentiable functions (max, min) say, Odyssée.

2.3.5 Hybrid method

The weakness of genetic algorithms (GAs) comes from the fact that they require a large number of evaluations of the objective function (fitness) which is very time consuming especially in CFD, (where the solution of the PDE is necessary.) Therefore a local optimization technique is added to avoid this lack of efficiency.

1. Define each of the individual members of the population:

Considering a simple 2-dimensional case, for instance, the geometry of a 2-d airfoil is to be defined. In this case a B-spline [18] could be used. A spline is defined by the vertices of its control polygon. By taking x-position coordinates of the vertices as fixed, an array which stores the y-coordinates can encode the geometry (shape), see [17].

2. Choose the initial population:

Each solution (shape) will be represented by one array (chromosome). The typical GA will generate the initial population randomly. It would

be useful to add some good candidates into the initial population, obtained by the gradient-based method.

3. Evaluate the fitness:

Evaluate the fitness of each airfoil expressed by the different arrays.

The fitness is totally dependent on the objective function.

4. Rank and select chromosomes corresponding to fitter airfoils for reproduction.

5. Apply operators of mutation and crossover to generate new individuals.

6. The population is replaced by the offspring .

Usually the task is to avoid ‘pre-mature’ convergence (converging too fast to find the global optimum), an important issue in GAs. The idea presented above includes some local optimum which might cause this effect. An alternative choice is to use the gradient-based methods to improve the best individual only at each iteration, see [41].

2.4 Shape parameterization

Shape representation is important in shape design. In this subject, how to choose the design parameters remains an open issue. There are two main branches:

- Shape parameterization
- Mesh-based

In the latter branch, each node of the mesh becomes the design variable which will obviously lead to a huge design space. However, the computational cost will not necessarily increase heavily if the adjoint method or the reverse mode of automatic differentiation is in use. But discontinuities may occur on gradients which require a smooth technique.

For shape parameterization, the basic idea of this representation is to assign a coefficient to each of the shape functions:

$$\Gamma = \sum_{i=1}^n \alpha_i f_i(\mathbf{x}). \quad (2.50)$$

In the above equation, Γ is the geometry required to be represented, $f_i(\mathbf{x})$ are shape functions, α_i are real-valued coefficients corresponding to these shape functions. Obviously an assumption is made that the optimal shape can be represented by these shape functions, (sometimes called reference shapes).

The choice of this shape function can be a polynomial function, triangle function, or B-spline, Bézier function. For simplicity, in a 2-dimensional case, the equation below is given to represent the geometry:

$$y(\Gamma) = \sum_{i=1}^n \alpha_i f_i(x) \quad (2.51)$$

where y, x are the y, x coordinates of the shape to be represented in Cartesian space.

Generally speaking, not all of the functions can be chosen as shape functions; the shape functions must meet some requirements introduced by problems and have some features which help the user control the shape easily. One simple choice of shape functions is suggested by Hicks and Henne [24] (used in airfoil design); that is:

$$f(x) = \{\sin[\pi x^{\log \frac{1}{2}/\log t_1}]\}^{t_2} \quad 0 \leq x \leq 1 \quad (2.52)$$

It can be easily seen that the parameter t_1 controls where the maximum is while the parameter t_2 controls the width.

Bézier and B-splines are also used to parameterize the geometry. In this report functional B-splines are used to parameterize the geometry. For a Bézier curve over a general interval $[a, b]$, the shape function is defined by the following equation:

$$p(u) = \sum_{i=0}^m b_i B_i^m(u), \quad B_i^m(u) = \binom{m}{i} \frac{(u-a)^i (b-u)^{m-i}}{(b-a)^m}, \quad u \in [a, b].$$

The Bézier curve has the following properties:

- *Affine invariance*: Bézier curves are invariant under affine maps. Becomes more and more popular in shape representation.
- *Convex hull property*: A Bézier curve lies within the convex hull of its control polygon.
- *Endpoint interpolation*: A Bézier curve interpolates the first and last control points,

Although the Bézier curve has the above useful advantages, it also has a big limitation. This is that the number of control points will decide the order of the curve. This limitation can be avoided by another shape representation technique; B-splines:

$$y(u) = \sum_{i=1}^n d_i N_{i,k}(u),$$

where n is the number of the control points in the B-spline curve, d_i is the i -th control point, and the B-spline basis function is defined as:

$$\begin{aligned} N_{i,1}(u) &= 1 & \text{if } u_i \leq u < u_{i+1} \\ N_{i,1}(u) &= 0 & \text{otherwise} \\ N_{i,k}(u) &= \frac{u - u_i}{u_{i+k-1} - u_i} N_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} N_{i+1,k-1}(u) \end{aligned} \quad (2.53)$$

The knot set is $u_1, u_2, \dots, u_{n+k}$, and the order is k . The x coordinate of the control point is computing by the average knot in the functional B-spline case.

The B-spline has the following properties:

- *convex hull property*: A B-spline curve is contained in the convex hull of its control polyline.
- *local modification scheme*: Changing the position of control point P_i only affects the curve $C(u)$ on interval $[u_i, u_{(i+p+1)}]$.
- *affine invariance*
- *variation diminishing property* If the curve is in a plane, this means no straight line intersects a B-spline curve more times than it intersects

the curve's control polyline.

Besides all of these properties, an important advantage of the B-spline is that NURBs (Non-Uniform Rational B-splines) is a standard for geometry representation techniques in CAD systems.

B-spline surfaces can be regarded as a tensor product generalization of B-spline curves. A functional B-spline surface of order k by l takes the form

$$z(u, v) = \sum_{i=1}^n \sum_{j=1}^m d_{ij} N_{i,k}(u) N_{j,l}(v),$$

where associated with z is a knot set in u , $(u_i)_{i=1}^{n+k}$ and a knot set in v , $(v_j)_{j=1}^{m+l}$.

B-splines still have limitations in shape optimization, i.e. B-spline is not so good at representing non-smooth geometry. From the above content and based on the knot set knowledge of B-splines [18], it is true that B-spline can parameterize a geometry with singularity, if the multiplicity of the knot equals to the order. However, this is not a good geometry representation method because the number of control points will inevitably increase which makes a piecewise B-spline. In that case it would be a choice to use 2 B-splines rather one with singularity and more control points.

2.5 PDE solution

Solving the PDE is necessary in shape design because the objective functions in shape design problems are involved with the PDE. To solve the PDE, the

discretization is first applied. Since the method of discretizing the PDE is usually mesh based, a mesh or grid must be constructed first. The discretization of a PDE can be divided into two categories:

- temporal discretization
- spatial discretization

In practice, temporal discretization always uses the finite difference method, while spatial discretization varies by using different methods, such as FEM (Finite Element Method), FDA (Finite Difference Analysis), FV (Finite Volume Method), even SPH (Smoothed Particle Hydrodynamics, a gridless method).

After discretization, a system of algebraic equations is usually produced. Some methods have been presented to solve the system but no method has been proven to be superior to others in general.

2.5.1 The mesh

Mesh generation is an essential aspect of all numerical methods that employ finite differences, finite volumes and finite elements for the solution of partial differential equations (PDEs). In short, it consists in dividing bounded or unbounded flow domains into elements (triangles, quadrilaterals, polygons in 2-D; tetrahedrons, parallelepipeds, n-edrons, etc.) called cells.

There are two classes of meshes in the geometry space, namely

- structured mesh

- unstructured mesh

Structured mesh and unstructured mesh are different in the way the mesh points are locally organized. If the local organization of the mesh points and the form of the elements are defined by a general rule, the mesh is a structured mesh, otherwise, it is an unstructured mesh

The advantages of using structured mesh are listed as follows:

- it can be used in conjunction with some very efficient PDE solving methods, such as the multigrid method
- less memory storage
- simple and fast to generate the mesh

However, the structured grid is not suitable for a domain with a very complicated geometric boundary; unfortunately this is true in most of the physical applications.

An unstructured grid has better geometric flexibility than a structured mesh. It can be generated faster on most complex domains and exists for all domains. Furthermore mesh refinement can be done without difficulty. However, the storage of the unstructured mesh data is not easy (it requires information on which node is neighbour to which), and also it takes far more memory than in a structured mesh does, and thereby hindering parallelization of the computer code.

The Advancing Front technique and Delaunay Triangulation are the two most popular methods to generate the mesh. Below is a brief introduction to these two methods.

1. *Advancing Front Technique.* In this technique mesh elements are placed at the boundary and advance towards the interior of the domain. The first triangle consists of the contiguous points closest to a corner point. Further triangles are constructed by connecting interior points to the existing front line of triangles.
2. *Delaunay Triangulation.* A Delaunay triangulation for a set P of points in the plane is a triangulation such that no point in P is inside the circumcircle of any triangle in the triangulation. The Delaunay triangulation procedure looks like the following: a central interior point is connected to a neighbour point by a straight line. Another line is constructed which is perpendicular to the first one, and half-way between the points. If the same procedure is performed four times between the current central point and the other connected points, then the perpendicular lines form a pentagon. Triangles are formed by connecting the points which have common boundaries of pentagons. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid "sliver" triangles. Delaunay triangulations are often used to create meshes for the finite element method, because of the angle guarantee and its efficiency on mesh triangulation.

The initial mesh used to solve the PDE may not be suitable to produce a sufficiently accurate solution. In that case, an improved mesh is therefore adopted by refining either uniformly or some specified area. These are called global mesh refinements and local mesh refinements.

If a certain refinement operation is applied to all the elements in the domain, then this procedure is regarded as a global mesh refinement. If the refinement operation only needs to be performed on some regions within the domain, then it is referred to as local mesh refinement.

The purpose of using mesh refinement is to achieve a sufficiently accurate approximation of the solution. Furthermore, it may also save computational costs. The mesh refinement approach which diminishes the mesh size by inserting extra nodes is called h-refinement. Another mesh refinement called p-refinement which increases the polynomial order of the basis function (in FEA methods) or through the usage of higher order stencils (in FDA methods), means more nodes are involved in every equation. Local mesh refinement is preferable to global mesh refinement because the latter could cause inefficiency, see [51].

Both structured and unstructured meshes can be locally refined, but the mesh refinement is usually applied to the unstructured mesh in practice. The reason for this is not only because of the memory storage, but also because of the hanging node in the structured mesh, which means a node is a vertex of one element but is not a vertex of the adjacent element.

2.5.2 Finite difference method

This section presents a quick introduction to the finite difference method. This method can be used to represent an approximation to the derivative terms of the partial differential equation (PDE) at the discrete points.

Finite difference methods can be classified into two groups:

- Implicit method: The PDE can be solved indirectly by solving a system of simultaneous linear equations.
- Explicit method: The PDE can be solved directly using the appropriated boundary conditions, proceeding backwards in time through small intervals and a finite difference operator to the discrete point.

The finite difference method is usually connected with a structured mesh. In a 2-dimensional case, for a given continuous function $\phi(x, y)$, the approximation for this function at the discrete point $(x + \Delta x, y + \Delta y)$ can be described by using the Taylor series expansions around its adjacent points (x, y) .

$$\begin{aligned}\phi(x + \Delta x, y + \Delta y) &= \phi(x, y) + \frac{1}{1!}(\Delta x \frac{\partial}{\partial x} + \Delta y \frac{\partial}{\partial y})\phi(x, y) \\ &+ \frac{1}{2!}(\Delta x \frac{\partial}{\partial x} + \Delta y \frac{\partial}{\partial y})^2\phi(x, y) + \dots + O(\Delta x^{n+1}, \Delta y^{n+1}) \\ &= \phi(x, y)(\Delta x \frac{\partial \phi}{\partial x} + \Delta y \frac{\partial \phi}{\partial y}) + \frac{1}{2}(\Delta x^2 \frac{\partial^2 \phi}{\partial x^2} + \Delta y^2 \frac{\partial^2 \phi}{\partial y^2} \\ &+ 2\Delta x \Delta y \frac{\partial^2 \phi}{\partial x \partial y}) + \dots + O(\Delta x^{n+1}, \Delta y^{n+1})\end{aligned}$$

In a three dimensional case, the approximation is given by the following equation:

$$\begin{aligned}\phi(x + \Delta x, y + \Delta y, z + \Delta z) &= \phi(x, y, z) + \frac{1}{1!}(\Delta x \frac{\partial}{\partial x} + \Delta y \frac{\partial}{\partial y} + \Delta z \frac{\partial}{\partial z})\phi(x, y, z) \\ &+ \frac{1}{2!}(\Delta x \frac{\partial}{\partial x} + \Delta y \frac{\partial}{\partial y} + \Delta z \frac{\partial}{\partial z})^2\phi(x, y, z) \\ &+ \dots + O(\Delta x^{n+1}, \Delta y^{n+1}, \Delta z^{n+1})\end{aligned}$$

In practice, the number n cannot be infinite. If the Taylor expansion terms where $n > p$ are cut off from the approximation, then it has a truncation error given by the following expression:

$$\frac{1}{(p+1)!}(\Delta x \frac{\partial}{\partial x} + \Delta y \frac{\partial}{\partial y} + \Delta z \frac{\partial}{\partial z})^{(p+1)}\phi(x, y, z) + \dots + O(\Delta x^{n+1}, \Delta y^{n+1}, \Delta z^{n+1})$$

where the first term $\frac{1}{(p+1)!}(\Delta x \frac{\partial}{\partial x} + \Delta y \frac{\partial}{\partial y} + \Delta z \frac{\partial}{\partial z})^{(p+1)}\phi(x, y, z)$ is the so called the principle part of the truncation error.

To approximate the first order derivatives, the finite difference method has the forward difference, the backward difference and the central difference operators. To approximate the second order derivatives, higher order finite difference operators are usually employed, which normally means more grid points are involved.

The finite difference method is a straightforward way to approximate the PDEs. The derivative terms in the PDE can be directly replaced by using the finite difference method while the FEM requires integral equations. Fur-

thermore, the boundary conditions can be implemented in a straightforward manner, where the FEM requires a proper shape function which can satisfy the boundary condition. However, FDM has difficulties on complex domains, which raises the FEM.

2.5.3 Finite Element Method

Finite element methods are used for finding approximate solutions to a physical problem defined in a domain. The method is based on the technique of dividing the region or domain of the problem into a number of finite elements and then finding the best possible solution which is continuous outside the elements. The accuracy of the solution is mainly dependent on the number and size of the elements and on the types of approximation function used within the elements.

In a typical finite element problem the solution is usually derived in four separate phases in the following order:

- formulation phase

The physical region of interest is divided into elements and element types are defined. The approximation to the exact solution at each element is presented by appropriate interpolation functions.

- evaluation and assembly phase

The contribution of each element to the problem formulation is computed. This usually raises the stiffness matrix and force vectors. The contributions from the individual elements are assembled to produce a

large system of equations for solution.

- solution phase

These equations are solved to find the primary unknowns of the problem. These unknowns are generally values of some physical quantity at the nodes of the elements.

For example: to solve a ordinary differential equation in 1-dimension:

$$\frac{\partial \Phi}{\partial x} = f(x) \quad (2.54)$$

the unknown solution $\phi(x)$ can be approximated by the following equation:

$$\Phi(x) = \alpha_1 N_1(x) + \alpha_2 N_2(x) + \dots = \sum \alpha_i N_i(x) \quad (2.55)$$

where α_i are unknown coefficients and $N_i(x)$ are the shape functions. The residual is given by the following equation:

$$R(\phi, x) = \frac{\partial \Phi}{\partial x} - f(x) \quad (2.56)$$

The finite element method will try to solve the following equation:

$$\int_{\Omega} W(x) R(\Phi, x) d\Omega = 0 \quad (2.57)$$

where the $W(x)$ is called the weight function, and this equation is applicable for one dimensional case. Extension into more dimensions can be obtained by following the same strategy used before. The weight function is introduced in order to be able to minimise the residual at each element. The choice of

the weight function depends on the applied scheme. The approach where the weight function is identical to the shape function is called the Galerkin method ($W_i = N_i$).

The difference between the FEM and the FDM is that FEM approximates the solution of the PDE, while FDM is an approximation of the PDE.

2.5.4 Subspace method

The Krylov subspace method is the most important class of iterative methods. It is a projection method on Krylov subspaces. For solving the system equation

$$A\mathbf{x} = \mathbf{b},$$

the projection methods seek an approximation solution, $\mathbf{x}_k$, from an affine subspace $\mathbf{x}_c + K_k$ by imposing the Petrov-Galerkin condition:

$$\mathbf{b} - A\mathbf{x}_k \text{ is perpendicular to } L_k,$$

where L_k is another subspace of dimension k and x_c is an arbitrary initial approximation. The choices of subspace L are many; Krylov subspace methods choose

$$K_k(A, r_c) = \text{span}\{\mathbf{r}_c, A\mathbf{r}_c, A^2\mathbf{r}_c, \dots, A^{k-1}\mathbf{r}_c\}$$

where $\mathbf{r}_c = \mathbf{b} - A\mathbf{x}_c$.

The approximations obtained are of the form

$$A^{-1}b \approx \mathbf{x}_k = \mathbf{x}_c + \mathbf{q}_{k-1}(A)r_c,$$

where $\mathbf{q}_{k-1}$ is a polynomial of degree $k - 1$. If the initial guess is chosen as $\mathbf{x} = 0$ the following simple equation is achieved:

$$A^{-1}b \approx \mathbf{x}_k = \mathbf{q}_{k-1}(A)\mathbf{r}_c.$$

Many variants exist, depending on the subspace L . Two choices of L_k are given below:

- $L_k = K_k$
- $L_k = AK_k$.

3 The Super-Reduced method

In this new method, the geometry is parameterized by B-spline shape functions with nominal uniform knot set. Therefore, the control points of the B-spline curves become the design variables in the optimization scheme. The knot insertion algorithm has been firstly introduced to keep the geometry unchanged whilst increasing the number of control points. The initial number of control points can be set to a small number, potentially decreasing the number of optimization iterations. To achieve a suitable accuracy the final number of control points can be very large for the value of the objective function. This could be very useful especially when the objective function is sensitive to the geometry change. In the approach taken here the knot insertion algorithm starts when the solution is close to the optimum solution. After that, a large number of design variables will be produced as specified by the user. Since the knot insertion algorithm does not change the geometry, the optimization problem will be kept the same.

Beside this core idea, the super-reduced idea and mesh refinement are also incorporated. The super-reduced idea is employed to convert the constrained problem into an unconstrained one (where the constraints are kept unchanged all the time). Using this method the gradient becomes a super-reduced gradient. The applications of the super-reduced idea can be also found in [62], which uses Euler equation as the governing equation. In this thesis, using the super-reduced idea is an independent research, and the idea has been applied into different applications. Furthermore, the validation of the super-reduced

idea is also explored

Mesh refinement is extensively used in shape optimization. This approach is very useful when the exact gradient is difficult to obtain. In doing so, most of iterations will be done on the coarse grid and with relatively few on the refined grid, thus making the method more efficient.

The developed new method is the first method which employs the knot insertion idea, the super-reduced idea and mesh refinement.

To solve a general shape optimization problem, an optimization scheme is necessary. The new method, which is BFGS-based, is used because of its super linear convergence and better efficiency. The selection of the search direction and the update of approximate Hessian matrix follows the same rule as the BFGS method [20]:

$$\mathbf{s}_k = -H(\mathbf{x}_k)^{-1} \mathbf{g}_k \quad (3.1)$$

$$H_{k+1} = H_k + \frac{\mathbf{q}_k \mathbf{q}_k^T}{\mathbf{q}_k^T \mathbf{p}_k} - \frac{H_k \mathbf{p}_k \mathbf{p}_k^T H_k}{\mathbf{p}_k^T H_k \mathbf{p}_k} \quad (3.2)$$

where $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{q}_k = \mathbf{g}(\mathbf{x}_{k+1}) - \mathbf{g}_k$, and $\mathbf{x}_k, \mathbf{g}_k$ denotes the point and the gradient at the k -th iteration respectively.

Using any gradient-based method requires the user to supply the design variables. In the new method, B-spline is chosen to parameterize the shape, therefore the objective function becomes a function of the control points of

the B-spline curves, denote by $F(\alpha_1, \alpha_2, \dots, \alpha_n)$, where $(\alpha_1, \alpha_2, \dots, \alpha_n)$ is a set of control points to describe the geometry $\partial\Omega$.

The number of control points, n , for the initial shape can be chosen as small as possible. The method first inputs control points as design variables to generate the shape. Based on the shape, the value of the objective function and the gradient is computed. The optimizer will repeat this process until a certain criteria is reached.

The criteria is chosen as

$$\|F_{k+1} - F_k\| < \eta$$

while η is a sufficiently small value to demonstrate whether the value is close to the optimum value. Once this criteria is met, the knot insertion algorithm and the mesh refinement start. The knot insertion algorithm [8] insert numbers of knots to the knot set and corresponding numbers of control points while the geometry is not changed. In that case, the numbers of control points are increased from n to m where $m > n$. Suppose

$$(\alpha_1, \alpha_2, \dots, \alpha_n)$$

are the control points which define the geometry before knot insertion, while

$$(\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_m)$$

are the control points which define the geometry after knot insertion. Then, the following equation is obtained:

$$\partial\Omega_1(\alpha_1, \alpha_2, \dots, \alpha_n) = \partial\Omega_2(\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_m) \quad m > n \quad (3.3)$$

Therefore the knot insertion algorithm does not change the optimization because the objective function is a function of the geometry which is kept unchanged.

$$F(\partial\Omega_1) = F(\partial\Omega_2) \quad (3.4)$$

Introducing a knot insertion algorithm is the core idea of the new method. Besides this, the new method includes mesh refinement, and a super-reduced technique. These will be discussed in the following section. The procedure for using the new method will be described thereafter. The method, and finally its advantages, will be addressed.

3.1 Knot insertion algorithm

A knot insertion algorithm [18] can insert a knot into a B-spline curve without changing its shape. After knot insertion, the B-spline function ($\mathbf{p}(u)$) changes from

$$y(u) = \sum_{i=1}^n d_i N_{i,k}(u), \text{ with knot set } (u_i)_{i=1}^{n+k} \quad (3.5)$$

to

$$y(u) = \sum_{i=1}^{n+1} \hat{d}_i N_{i,k}(u), \text{ with knot set } (u_i^1)_{i=1}^{n+1+k}. \quad (3.6)$$

where k is the order, u is the parameter value, d_i is the i -th control point for the B-spline curve before knot insertion and $\hat{d}_i$ the i -th control point for the B-spline curve after knot insertion. The second knot set differs from the first one in that it has an additional knot inserted somewhere in the interval $[u_k, u_{n+1}]$. This interval is selected because $u_1 = u_2 = \dots = u_k$, $u_{n+1} = u_{n+2} = \dots = u_{n+k}$ and we want to keep this property to let the first control and the last control points in the B-spline interpolate the first data point and the last data point. Thus the knot can be inserted in the interval $[u_k, u_{n+1}]$ only.

If a knot $\hat{u}$ is to be inserted into the knot set, coinciding with the knot u_{p+1} , say, which already has multiplicity s (if it does not already occur in the knot sequence, then $s = 0$), and the new knot set is denoted as

$$u_i^1 = \begin{cases} u_i & i \leq p \\ \hat{u} = u_{p+1} & i = p + 1 \\ u_{i-1} & i \geq p + 2, \end{cases} \quad (3.7)$$

then the number of control points has to increase from n to $n + 1$ because the number of the previous knot set increases from $n + k$ to $n + k + 1$.

Since the B-spline has a local modification property, only some of the control points need to change by a linear interpolation of the previous control points;

that is:

$$\hat{d}_i = \alpha_i^1 d_i + (1 - \alpha_i^1) d_{i-1} \quad i = p - k + s + 2, p - k + s + 3, \dots, p \quad (3.8)$$

where $\alpha_i^1 = (\hat{u} - u_i^1)/(u_{i+k}^1 - u_i^1) = (\hat{u} - u_i)/(u_{i+k-1} - u_i)$. This is the core idea of the knot insertion algorithm, details of which can be found in [18].

The knot insertion algorithm is as follows:

Algorithm: single knot insertion

```

for i = 1, p - k + s + 1, do:
     $\hat{d}_i = d_i$ 
for i = p - k + s + 2, p, do:
     $\alpha_i^1 = (\hat{u} - u_i^1)/(u_{i+k}^1 - u_i^1) = (\hat{u} - u_i)/(u_{i+k-1} - u_i)$ 
     $\hat{d}_i = \alpha_i^1 d_i + (1 - \alpha_i^1) d_{i-1}$ 
for i = p + 1, n + 1, do:
     $\hat{d}_i = d_{i-1}$ 

```

3.1.1 Use of knot insertion algorithm

The knot insertion algorithm allows the user to add additional control points, i.e. design variables in optimization, to the B-spline curve without changing the geometry. Therefore, it is possible for the user to start the optimization

iteration with only a very small number of control points, while ending with a large number of design variables to describe the geometry. The purpose of using the knot insertion algorithm is to decrease the numbers of optimization iterations while obtaining a relatively better value of the objective function.

We use knot insertion algorithm by first starting with a few control points (as few as possible), then, when the code finds it is close to the optimal value, the knot insertion algorithm starts. An example is given below to show how we insert the knots.

Given an initial knot set, $(u_i)_{i=1}^{n+k}$:

$$(0, 0, \dots, 0, \underbrace{\frac{1}{n-k+1}, \frac{2}{n-k+1}, \frac{3}{n-k+1}, \dots, \frac{n-k}{n-k+1}}_{n-k}, \underbrace{1, 1, \dots, 1}_k)$$

the knot set after insertion is $(u_i^1)_{i=1}^{2n+1}$:

$$(0, 0, \dots, 0, \underbrace{\frac{1}{2*(n-k+1)}, \frac{2}{2*(n-k+1)}, \dots, \frac{2*(n-k)+1}{2*(n-k+1)}}_{2*(n-k)+1}, \underbrace{1, 1, \dots, 1}_k)$$

After this knot insertion it will have $2 * n - k + 1$ control points with the shape unchanged if the above procedure is followed. By repeating the knot insertion algorithm the control points will increase as required.

3.2 Mesh refinement

Starting mesh refinement at a certain stage in the optimization loop is not a new idea. The purpose of doing this is to make the optimizer do most of the iterations on the coarse mesh and relatively few on the refined mesh. To refine the mesh, there are two choices: local refinement and global refinement. Since the global refinement may introduce inefficiency [51], local refinement is used in this method.

Two important questions are how coarse the initial mesh should be and how refined the final mesh should be. To answer the first question, let us recall Taylor's expansion:

$$F(\mathbf{x}_k + \alpha_k \mathbf{s}_k) \approx F(\mathbf{x}_k) + \alpha_k [\mathbf{g}]^T \mathbf{s}_k + \frac{1}{2} \alpha_k^2 [\mathbf{s}_k]^T H \mathbf{s}_k, \quad (3.9)$$

where α_k , a positive real number, is the step size in the search direction $\mathbf{s}_k$, $\mathbf{g}$, H denotes the analytic gradient and analytic Hessian matrix respectively, and $\mathbf{s}_k = -H(\mathbf{x}_k)^{-1} \mathbf{g}(\mathbf{x}_k)$. ($H(\mathbf{x}_k)$, $\mathbf{g}(\mathbf{x}_k)$ are approximated value of the Hessian matrix and the gradient for the user specified mesh.)

In order to obtain a decrease in the value of the objective function, the following condition:

$$[\mathbf{g}]^T \cdot \mathbf{s}_k < 0, \quad (3.10)$$

is necessary, i.e, if

$$[\mathbf{g}]^T \cdot H(\mathbf{x}_k)^{-1} \mathbf{g}^h < 0 \quad \text{where} \quad \mathbf{g}^h = \mathbf{g}(\mathbf{x}_k), \quad (3.11)$$

is satisfied, then it is guaranteed that the optimizer is able to find a smaller value than $F(\mathbf{x}_k)$.

In the above formula $\mathbf{g}^h$ represents the approximated gradient in a mesh where the mesh size is h , and $H(\mathbf{x}_k)$ is an approximated Hessian matrix by BFGS update, which is a positive definite matrix. The initial approximation for such a matrix is usually the unit matrix.

Concerning the refinement of the final mesh, one can use error analysis to find out how the error (mainly discretization error) influences the value of F , or one can use the consistent gradient theory [40], [46]. Error analysis is used in the application of this method.

3.3 Super-reduced idea to compute gradients

A given shape optimization problem is often a PDE-constrained optimization problem in the CFD area with or without other constraints. For a shape optimization with one equality constraint,

$$\min_{\alpha} F(\alpha, \Phi(\alpha)) \quad \text{subject to} \quad c(\alpha) = 0, \quad (3.12)$$

where $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$ is a column vector consisting of n numbers of design variables, $\Phi(\alpha)$ represents the solution to the partial differential equation, and $c(\alpha)$ is the constraint.

It is obvious that solving an unconstrained optimization problem is much easier than solving a corresponding constrained optimization problem. The super-reduced idea, see [62], is introduced to convert the constrained optimization problem into an unconstrained optimization problem. The idea is to first solve the PDE constraint in each optimization iteration to reduce the above optimization system and then to solve the constraint to super-reduce the system.

In the new method, a design variable, say α_i , is chosen and this design variable can be uniquely defined by other design variable, i.e.

$$\alpha_i = E(\alpha_1, \alpha_2, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n) \quad (3.13)$$

where E is a function of $(\alpha_1, \alpha_2, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n)$. In this case, α_i is a dependent design variable, while all other design variables are independent.

Therefore, the constrained optimization problem (Equation 3.12) becomes the following unconstrained optimization problem:

$$\min_{\hat{\alpha}} F(\alpha_1, \alpha_2, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n) \quad \hat{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n]^T \quad (3.14)$$

provided that the partial differential equation is solved in each optimization iteration.

The solution provided by solving the above system is equivalent to the solution provided by the Lagrange system below

$$\min L(\alpha_1, \alpha_2, \dots, \alpha_n, \lambda_1) = \min F(\alpha_1, \alpha_2, \dots, \alpha_n) - \lambda_1 c(\alpha_1, \alpha_2, \dots, \alpha_n) \quad (3.15)$$

as long as the selected dependent variable is uniquely defined by other design variables. The mathematical proof for this argument is simple and given in Appendix B.

3.4 The procedure for the new method

The procedure for the new method is presented as follows:

1. Start from the initial design point $\mathbf{x}_0$, the inverse matrix of approximated Hessian matrix, B_0 .
2. Evaluate the value of the objective function, and evaluate/update the gradient.
3. Set the search direction $\mathbf{s}_k = -B_k \mathbf{g}_k$, Check the magnitude of the search direction $\mathbf{s}_k$ to avoid potential self-intersected shape. Use the line search method to find the step size α and keep the constraint all the time.
4. Update the design point $\mathbf{x}_k \longrightarrow \mathbf{x}_{k+1}$. Evaluate gradient at the new

point $\mathbf{x}_{k+1}$.

5. Update the Hessian matrix.
6. Decide whether the mesh refinement and the knot insertion algorithm should be employed.
7. If the knot insertion algorithm is required, produce the new design point $\hat{\mathbf{x}}_k$, compute the value of the objective function and the gradient.
8. Go back to step 2 if the termination criteria are not satisfied, otherwise output the result.

3.5 The advantages for the new method

The advantages for the new method are discussed as follows:

- Using this method, a optimum shape with a good value of the objective function can be found efficiently. Generally using less design variables the optimizer will require less iterations to converge. However, the value of the objective function obtained may not be very good, particularly in the case where the value of the objective function is sensitive to the shape change. On the other hand, by using a large number of design variables the optimizer may require a large number of iterations to find a optimum shape. The new method can find a good value of the objective function with a smaller initial number of design variables and a fewer number of iterations.
- The new method shares the super linear convergence feature of the BFGS method. Furthermore, in the tested cases used, its efficiency for

finding a optimum shape is better than the BFGS method, as shown later.

- It makes the BFGS based methods possible for a large number of design variable problems. Generally the BFGS method is not a good choice in this case because of its computational cost for the matrix operations. The new method however will limit most of the matrix operations to those of small dimension.

3.6 Preventing self-intersecting shapes

Preventing self-intersected shapes is not an issue in most shape optimization problems. By making some preconditions or using different shape representation techniques, self-intersecting shapes can be avoided. For example, in [5] the authors use a Bézier curve to minimize the least square difference of the designed shape and the original shapes. Therefore, the designed shape will be close to the original shape, which is not self-intersecting since the geometry change is small.

An idea can be borrowed from airfoil design. The shape of a 2-dimensional airfoil can be represented by a camber line with an equation to describe the relationship among the coordinates of the camber line. Therefore, in 2-dimensional cases, the coordinates for the points in the upper curve (x_u, y_u) and the coordinates for the points in the lower curve (x_l, y_l) can be represented by the equation arrays stated below.

$$\begin{aligned}
x_u &= x - y_t(x) \sin(\alpha) \\
y_u &= y_c(x) + y_t(x) \cos(\alpha)
\end{aligned} \tag{3.16}$$

and

$$\begin{aligned}
x_l &= x + y_t(x) \sin(\alpha) \\
y_l &= y_c(x) - y_t(x) \cos(\alpha)
\end{aligned} \tag{3.17}$$

where $y_t(x)$ is the thickness function, $y_c(x)$ is the camber line function, and

$$\alpha = \tan^{-1} \left\{ \frac{dy_c}{dx} \right\}$$

is the camber line slope.

In the numerical test case, it is necessary to avoid a self-intersecting shape since all control points have the freedom to move anywhere in 2-dimensional case. Once the shape is self-intersecting, obviously the mesh general will fail to work and then the PDE solver will return 0 (in most cases) to the value of the objective function because no mesh is actually input to the PDE solver.

The idea of preventing self-intersecting shapes is quite simple and straightforward. We detect whether

$$Y_U > Y_L, \quad \text{if} \quad X_U = X_L$$

for any points $\mathbf{P}_U = (X_U, Y_U)$ in the upper curve and any points $\mathbf{P}_L(X_L, Y_L)$ in the lower curve. If this is true we know there is no self-intersecting shapes.

In the code, it is impossible to process an infinite numbers of points and also it is not necessary to detect an infinite numbers of points because the boundary is constructed by a series of straight lines. Therefore a finite numbers of points can be chosen, but all of the boundary points used in the mesh generator have to be included for comparison.

3.7 Algorithms

In order to explain the new algorithm in detail, it is compared with the typical BFGS method. The new method is designed to deal with the constrained optimization problems with equality constraints, but its behaviour is the same as an unconstrained optimization algorithm. It can be shown that the algorithm has the same advantage of the typical unstrained optimization, the super linear convergence ($\frac{\|\mathbf{x}_{k+1}-\mathbf{x}_*\|}{\|\mathbf{x}_k-\mathbf{x}_*\|} = c_1$ where $2 > c_1 > 1$, $0 < c_2 < 1$ and $\mathbf{x}_*$ refers to the optimum point).

Algorithm: algorithm of a typical BFGS method

- 1, Input start point $\mathbf{x}_0$, B_0 , termination criteria.
- 2, For given k , set the search direction $\mathbf{s}_k = -B_k \mathbf{g}_k$.
- 3, Compute a step size α by line search algorithms or the trust region method.
- 4, Compute the update matrix according to a given formula (BFGS).
- 5, Update the inverse matrix of approximate Hessian matrix.
- 6, Continue iteration $k = k + 1$ until termination criteria are satisfied.

In the above algorithm, B is the inverse matrix of the approximated Hessian matrix.

The new algorithm

The new algorithm is presented as follows:

- Input start point $\mathbf{x}_0$, B_0 , termination criteria. (B_0 can be chosen as the unit matrix or reading from a file.) Set iteration number (1) and refine flag (0).
- Evaluate value of the objective function by different subprograms, the value of the refine flag will decide which subprogram should be used.
- Evaluate the gradient when the iteration number equals to one

- Set the search direction $\mathbf{s}_k = -B_k \mathbf{g}_k$.
- Check the magnitude of the search direction $\mathbf{s}_k$ to avoid potential self-intersecting shapes.
- Use the line search program to find the step size α . The line search program is a special bisection method, it can keep the constraint unchanged.
- Update the point $\mathbf{x}_k \longrightarrow \mathbf{x}_{k+1}$ while keeping the constraints unchanged.
- Evaluate the gradient at the new point $\mathbf{x}_{k+1}$ by different subprograms, the value of the refine flag will decide which subprogram should be used.
- Check whether the shape is going to be self-intersecting. If not, exchange the last point ($\mathbf{x}_k$) and the new point ($\mathbf{x}_{k+1}$), exchange the new gradient vector and the previous gradient vector. If it is, go to the last point and treat this case with a self-intersection subprogram.
- Decide whether it is necessary to change the value of the refine flag.
- Continue with next k until the termination criteria are satisfied.

The new algorithm is complicated, but it is very easy to catch the main idea. The core idea of the new algorithm is to introduce the knot insertion algorithm, and to use the super-reduced-method to convert the constrained optimization problem into an unconstrained one by operating on one special point. This special point can be chosen as the point which has the maximum absolute value. Furthermore, in order to decrease the noise on the gradient

value we add mesh refinement into the code. The code will automatically use the refined subprogram when the solution is close to the optimum. These two features will not influence the behaviour of the BFGS method at all. If the shape is not going to self-intersect, then the self-intersecting shape programs will not operate. If this does occur, it will deal with the previous point rather than the current point.

3.8 The choice of the end condition

For an unconstrained optimization problem, the purpose is to find one optimal point where the gradient equals zero. For a constrained optimization problem, the situation becomes more complicated. However, it still aims to find the feasible optimal point, or KT point (Kuhn and Tucker, 1951), see [20].

In the geometry optimization, the accuracy of the gradient mainly depends on how refined the mesh is. Obviously it is difficult to produce an arbitrary refinement considering the computational cost and the memory storage, especially in a 3-dimensional case. Using mesh refinement can partially solve the problem, but it is still difficult to find a solution which is arbitrarily close to the analytical solution. Furthermore, from a practical point of view, machine precision must be considered.

There are some common conditions which can be used as terminal conditions for most optimization problems:

- the length of the step size is too small
- the maximum number of iterations is reached
- the convergence is reached

The first two terminal conditions are clear. Normally these two conditions indicate that the optimizer fails to find the optimal solution. The third one can be indicated by any of the following conditions:

1. $\|F_{k+1} - F_k\| \leq \epsilon$
2. $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq \epsilon^{1/2}$
3. $\|\mathbf{g}_k\| \leq \epsilon$
4. $\mathbf{x}_k^T H \mathbf{x}_k \leq \epsilon^{1/3}$

First of all, the value of ϵ cannot be smaller than the machine precision, this is a requirement for numerical convergence.

$|F_{k+1} - F_k| \leq \epsilon$ is usually an indicator for the descent of the objective function. The condition requiring $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq \epsilon$ has the corresponding physical meaning that the solution converges to the optimal shape, where the first condition $|F_{k+1} - F_k| \leq \epsilon$ requires that the value converges to the optimal value.

The condition requiring $\|\mathbf{g}_k\| \leq \epsilon$ is in agreement with the theory; this condition can be combined with the first condition. A combined condition will be stronger than any single ones.

A constrained optimization case requires the point to be feasible and also that the Lagrange multiplier are valid, i.e. that the current point is inside the feasible space and all the values of λ are positive or null.

3.9 Additional comments on the new method

The choice of the selected control points which is used to keep the constraint unchanged is not a issue in other super-reduced applications. But in this new method it is a problem because the initial and final number of design variables are significantly different.

It is highly recommended to be consistent in selecting the special control point which is used to keep the constraints preserved. Suppose

$$(\alpha_1, \alpha_2, \dots, \alpha_n)$$

are the control points which define the geometry before knot insertion, while

$$(\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_m)$$

are the control points which define the geometry after knot insertion. If α_n is selected as the initial special control point, then $\hat{\alpha}_m$ should be selected as the final special control point for consistency. If the middle point $\alpha_{n/2}$ is selected as the initial special control point, then the middle point $\hat{\alpha}_{m/2}$ should be selected as the final special control point, and so on.

The choice of η which is used to decide whether the knot insertion should start. This value can be regarded as a non-vital issue for the method as long as η is not too small or too large from the gaining experience of this project.

3.10 Summary

A new method is presented to deal with shape optimization problems. In this method, the geometry is parameterized by a special shape function, B-splines with nominal knot set. Therefore, the control points of the B-spline curves become the design variables in the optimization scheme. The core idea of the method is to introduce the knot insertion algorithm which can keep the geometry unchanged while increasing the number of control points. Therefore, the initial number of control points can be decreased by as few as possible which will decrease the number of optimization iterations, and the final number of control points can be very large which can improve the accuracy level for the value of the objective function. In this method, the knot insertion algorithm starts when the solution is close to the optimum solution. After that, a large number of design variables will be produced as specified by the user. Since the knot insertion algorithm does not change the geometry, the optimization problem is still kept the same. This means that the relationship between the objective function and the geometry is unchanged, and also the geometry remains the same.

Besides this core idea, the method employs the super-reduced idea and the

mesh refinement. The super-reduced idea is used to get rid of the equality constraint. Therefore a constrained optimization problem is converted into an unconstrained optimization problem and the constraints are kept unchanged all the time.

Mesh refinement is heavily used in shape optimization methods. This idea is very important when the exact gradient is difficult to access. The purpose of doing so is to make the optimizer do most of iterations on the coarse grid and few on the refined grid.

The algorithm and its advantages have been discussed. The choice of whether to start the knot insertion algorithm has also been presented. The current way to start the knot insertion algorithm is by using a global method.

4 Numerical tests for the method in 2-dimensional cases

In this chapter results for the 2-dimensional shape optimization problem using the new method are reported. The first example is a PDE-constrained optimization problem with an equality geometry constraint. The Poisson equation is chosen as the governing equation because solving this equation is simpler than other non-linear PDEs. Furthermore, the solution procedure for shape optimization problems is basically the same, i.e. independent of the governing equation. The data and results for different objective functions, different tolerances placed on the magnitude of the gradient vector, and different initial shapes are given. To show the efficiency of the new method the result produced is compared with the result produced by the BFGS method. The idea of using a high order B-spline is investigated to find out whether it could possibly enhance the performance of the new method.

The second example used is a transonic airfoil design problem. The governing equation is a full potential equation with boundary layer analysis. The result produced by the new method are compared with the EXTREM method developed by Jacob [63], and the BFGS method. The multi-objective optimization is presented because the single objective optimization cannot easily converge to a shape which is satisfactory at all design points.

4.1 Application for using Poisson's equation as the governing equation

For the Poisson equation stated below

$$\nabla^2 \Phi = -1.0 \quad \text{in } \Omega \quad (4.1)$$

with the boundary condition

$$\Phi = 0 \quad \text{on } \partial\Omega$$

The purpose is to find an optimum shape to maximise the maximum value in the domain (Ω) :

$$\max_{\partial\Omega} \max_{\Omega} F \quad (4.2)$$

subject to a geometric constraint, here:

$$\text{Area of the domain, } \Omega = k, \quad k \text{ is a constant} \quad (4.3)$$

After reorganizing, the objective function is chosen as:

$$F = -\max_{\Omega} \Phi \quad (4.4)$$

and is subject to two constraints:

1. Area of the domain, $\Omega = k$, k is a constant.
2. The Poisson equation with the specified boundary condition. (The PDE constraint)

4.1.1 Technical requirements

In order to solve this problem, the items below are necessary:

- *Representation of the geometry*

Two cubic B-spline curves with nominal uniform knot set are chosen to parameterize the geometry. Therefore the objective function becomes a function of the control points and the control points become the design variables in this optimization scheme.

- *Mesh generator*

A mesh generator is needed to produce mesh information for any suitable geometry. BAMG [28] is chosen as the mesh generator. This generator cannot refine the geometry globally but can refine it locally.

- *Poisson solver*

A PDE solver is necessary to solve the Poisson equation with any acceptable meshes. The solver used is CRANDNS, see [7], [8]. It is a parallel Fortran code and has good performance.

- *Optimization scheme*

The optimization scheme is the core software in this problem. An optimization scheme has been developed specifically for this problem, more details about the scheme can be found in Chapter 3.

The flow chart below can explain how these items are used in the optimization scheme.

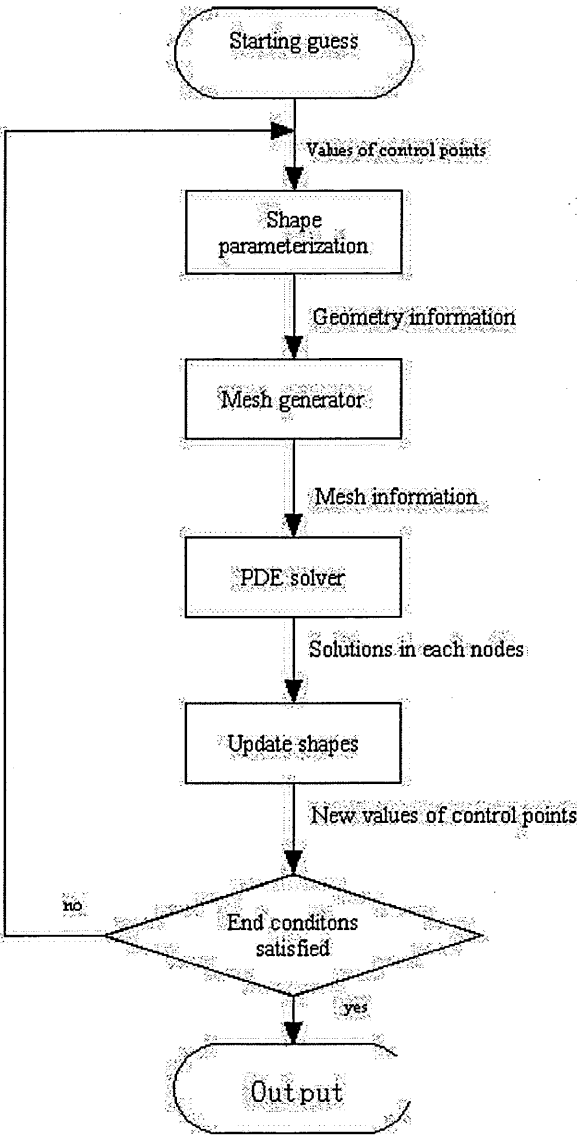


Figure 2: Typical procedure for solving shape optimization problems, which includes geometry representation, mesh generation, PDE solution, and optimization scheme

4.1.2 Theoretical results in shape representation

In this problem, B-splines are chosen to parameterize the geometry because:

- They are standard.
- They can represent complicated geometry with a low degree.
- They have built in smoothness if the knot set is chosen properly.

Two B-spline curves are chosen to parameterize the upper geometry and the lower geometry. See the example below:

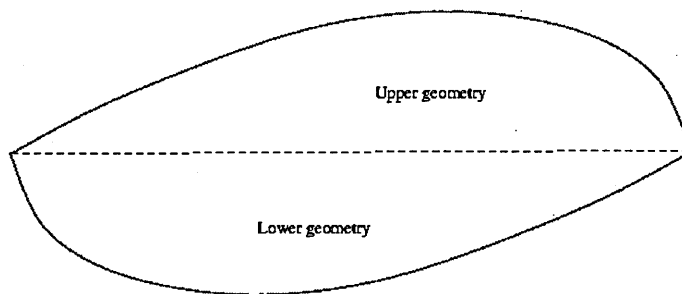


Figure 3: Sample splines to construct the closed geometry. Both of the upper curve and the lower curve are splines

We make the following assumption:

the distance between the first control point and the last point is fixed.

The reason for this assumption is that it is quite possible to obtain open shapes without it. There are some other ways to deal with this, but our method is a pragmatic way of dealing with this problem. Furthermore an extensive investigation on this assumption is given in Section 4.10

The expression for each of the two B-splines curves is given by the equation below:

$$y(u) = \sum_{l=1}^n d_l N_{l,k}(u) \quad 0 \leq u \leq 1 \quad (4.5)$$

where the B-spline basis function is defined as:

$$\begin{aligned} N_{l,1}(u) &= 1 & \text{if} & \quad u_l \leq u < u_{l+1} \\ N_{l,1}(u) &= 0 & \text{otherwise} \end{aligned}$$

$$N_{l,k}(u) = \frac{u - u_i}{u_{l+k-1} - u_l} N_{l,k-1}(u) + \frac{u_{l+k} - u}{u_{l+k} - u_{l+1}} N_{l+1,k-1}(u) \quad (4.6)$$

The knot set $u_1, u_2, \dots, u_{n+k}$ and the order k are not specified yet.

Suppose a knot set is chosen as below:

$$\underbrace{(0, 0, \dots, 0)}_k, \underbrace{\frac{1}{n-k+1}, \frac{2}{n-k+1}, \frac{3}{n-k+1}, \dots, \frac{n-k}{n-k+1}}_{n-k}, \underbrace{(1, 1, \dots, 1)}_k$$

In that case, the area of the shape is given by the equation below:

$$\begin{aligned} A(\Gamma) &= A(\Gamma^U) - A(\Gamma^L) \\ A(\Gamma^U) &= \int_0^1 (y^U(u)) du \\ A(\Gamma^L) &= \int_0^1 (y^L(u)) du \end{aligned} \quad (4.7)$$

provided that the shape consists of the upper geometry Γ^U and the lower geometry Γ^L are not crossed. In the above equation array, y^U, y^L denote the

upper boundary and lower boundary respectively, A^U , A^L denote the area of the upper part and the area of the lower part respectively. (See Fig 3)

Based on equation 4.7, the derivative of the area with respect to the control points in the upper curve is given by the equation below:

$$\frac{\partial A}{\partial d_i} = \int_0^1 N_{i,k}(u) du \quad (4.8)$$

where

$$\int_0^1 N_{i,k}(u) du = \frac{u_{i+k} - u_i}{k}. \quad (4.9)$$

and d_i refers to the i -th control point in the upper curve. Hence if a cubic B-spline curve with nominal uniform knot set is used, the gradient of the area with respect to the control points in the upper B-spline curve is given by

$$\frac{\partial A}{\partial d_i} = \frac{u_{i+4} - u_i}{4} \quad (4.10)$$

For the control points in the lower B-spline curve, the gradient is given by:

$$\frac{\partial A}{\partial d_j} = -\frac{u_{j+4} - u_j}{4} \quad (4.11)$$

where d_j refers to the j -th control point in the lower curve and the area is given by the following equation:

$$A = \sum_{i=1}^n \left(d_i \frac{u_{i+4} - u_i}{4} \right) - \sum_{j=1}^m \left(d_j \frac{u_{j+4} - u_j}{4} \right) \quad (4.12)$$

where n and m are the number of control points in the upper and lower curves respectively.

4.1.3 Compute the super-reduced gradient by using the finite difference method

Since the objective function F is a function of state variable Φ and the shape $\partial\Omega$, $(F(\Phi, \partial\Omega))$, and the relationship between Φ and $\partial\Omega$ is determined by the Poisson equation with specified boundary condition $(\Phi(\partial\Omega))$, the state equation (the Poisson equation) is solved first in each optimization iteration.

$$F(\Phi, \partial\Omega) \longrightarrow F(\partial\Omega)$$

Second, because $\partial\Omega$ is a function of design variables, i.e. control points of the B-spline curves, $\partial\Omega(d_1, d_2, \dots, d_n)$, $d_1, d_2, \dots, d_n$ are the control points of the B-spline curves, and the area of the domain needs to be fixed, one of the control points is moved in order to keep the area constant. In this case, the constraint equation $A = k$ needs to be solved as well.

Now the problem is redefined as:

$$F = F(d_1, d_2, \dots, d_{i-1}, d_{i+1}, \dots, d_n) \quad (4.13)$$

with d_i eliminated in each optimization iteration.

The justification for this method is given in the Appendix B. Based on the

proof, if d_i can be uniquely defined by $(d_1, d_2, \dots, d_{i-1}, d_{i+1}, \dots, d_n)$, then the optimal point produced by the super-reduced idea is also the optimal point for the Lagrange system.

The super-reduced gradient is computed by using the following finite difference method so that the constraints are kept unchanged:

$$\begin{aligned} \frac{\partial F}{\partial d_j} &\approx \frac{F(d_1, d_2, \dots, d_j + \epsilon, \dots, d_i + \delta, \dots, d_n) - F(d_1, d_2, d_3, \dots, d_n)}{\epsilon} \quad j \neq i \end{aligned} \quad (4.14)$$

where $j = 1, 2, \dots, n$, $j \neq i$ (if $j = i$, then $\frac{\partial F}{\partial d_i} = 0$) and δ is given by solving the constraints:

$$\text{Area of } \Omega(d_1, d_2, \dots, d_{i-1}, d_i + \epsilon, d_{i+1}, \dots, d_n + \delta) = \text{constant}.$$

This equation is actually a linear equation from equation 4.12.

4.1.4 Optimum shapes

The developed method has been used for varying initial shapes, different accuracy levels placed on the magnitude of gradient vector, different number of control points, and different objective functions. In all cases tested the solution converges, and the optimum shape is found.

Some optimum shapes are obtained without using mesh refinement. One of these is presented. The optimizer starts from the initial shape 1 (see Fig-

ure 4), and after 103 optimization iterations it finds the optimum shape (see Figure 5). The optimal value of objective function is -0.062202.

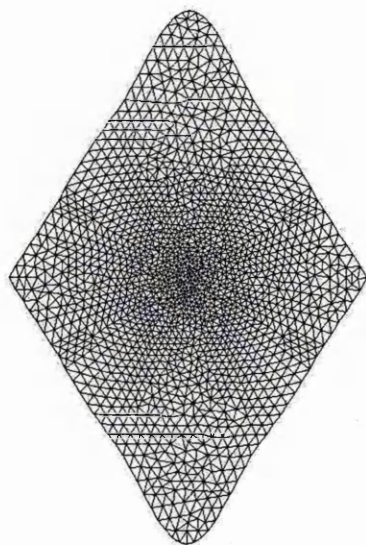


Figure 4: Initial shape 1

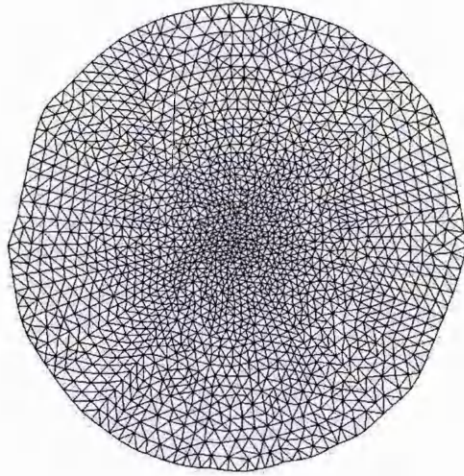


Figure 5: Optimum shape for initial shape 1 without using mesh refinement. Coarse mesh is in use with about 3300 elements

In this optimization, mesh refinement is excluded. The initial mesh has about 3300 triangles and the final mesh has nearly the same amount of triangles. The super-reduced idea is introduced in order to keep the constraint, so the area of the geometry does not change in the optimization.

All the results in the table below are produced with mesh refinement but without use of the knot insertion algorithm:

Starting guess	Tolerance	Iteration numbers	Value	Final shape
Fig. 4	7.0e-3	42	-0.062011092376	Fig. 6
Fig. 4	4.5e-3	51	-0.062091348827	Fig. 7
Fig. 4	3.5e-3	66	-0.062239391526	Fig. 8
Fig. 4	2.5e-3	94	-0.0623229957943	Fig. 9
Fig. 10	4.5e-3	62	-0.0621104533865	Fig. 11
Fig. 12	3.5e-3	48	-0.0622796812737	Fig. 13

Table 1: 2-dimensiional shape optimization using mesh refinement

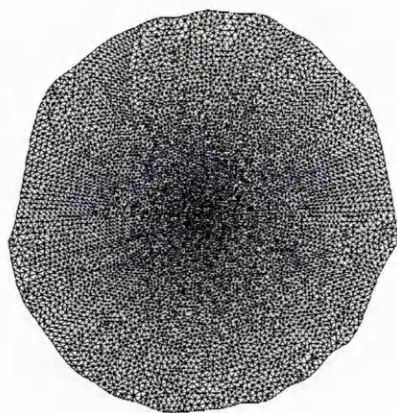


Figure 6: Optimum shape for initial shape 1 with $\|\mathbf{g}\| < 7.0e - 3$

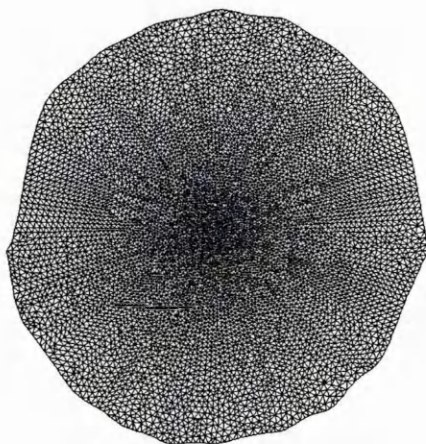


Figure 7: Optimum shape for initial shape 1 with $\|\mathbf{g}\| < 4.5e - 3$

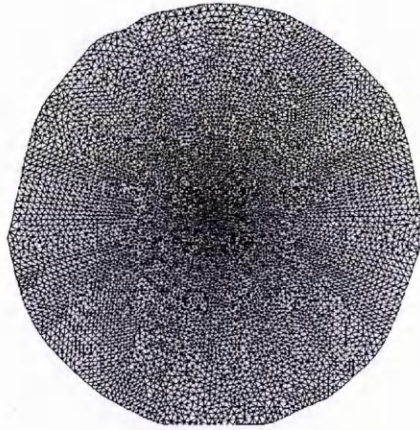


Figure 8: Optimum shape for initial shape 1 with $\|g\| < 3.5e - 3$

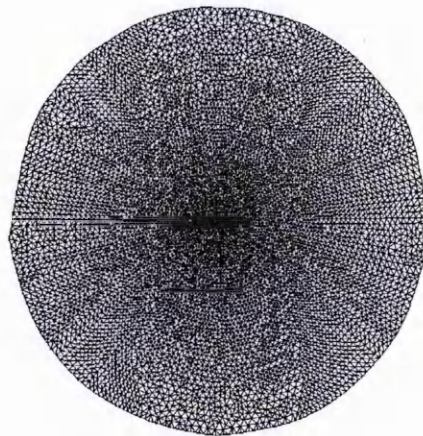


Figure 9: Optimum shape for initial shape 1 with $\|g\| < 2.5e - 3$

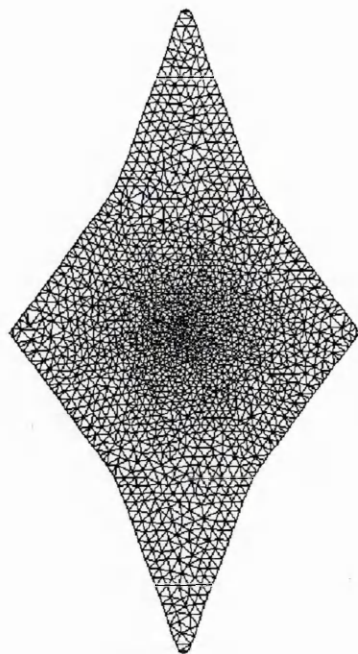


Figure 10: Initial shape 2

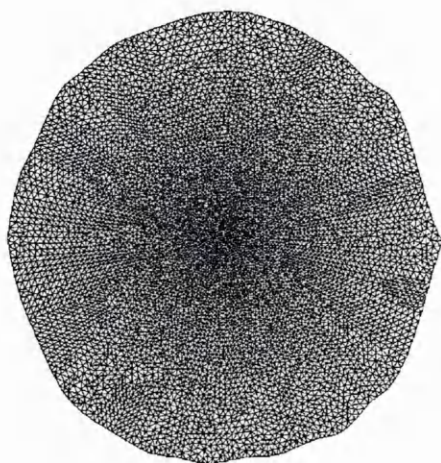


Figure 11: Optimum shape for initial shape 2 with $\|g\| < 4.5e - 3$

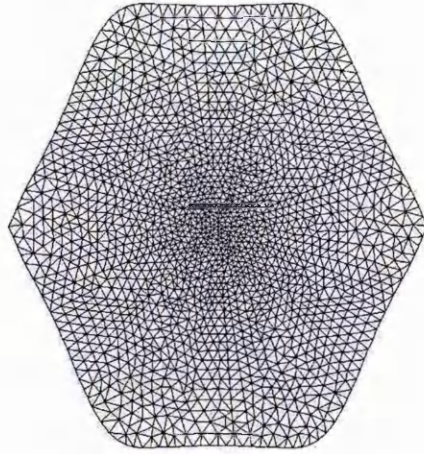


Figure 12: Initial shape 3

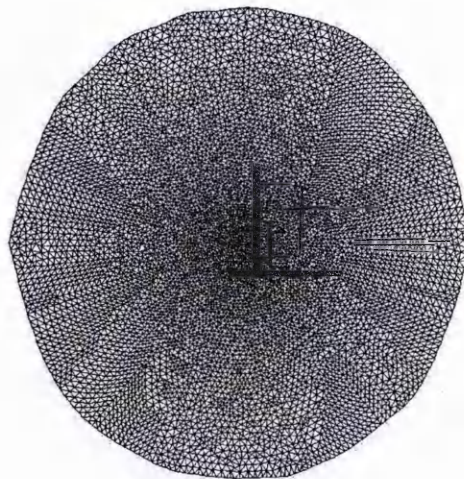


Figure 13: Optimum shape for initial shape 3 with $\|g\| < 3.5e - 3$

These optimum shapes are not the best shapes that the method can obtain. It can go further if the user requires. As long as the computational cost is

affordable, the results will approach closer and closer to the analytical solution. However, it is not a good idea to place too small a tolerance on the magnitude of the gradient vector. Of course it is still possible to find the optimum shape by mesh refinement but the computational cost may be huge. After all, in this design case there are 38 design variables and a PDE must be solved in each function call to the objective function. The error in the solution can be referred to section 4.4.1.

For these different initial shapes, the areas are not exactly the same. This means the area for the initial shape 1, the initial shape 2 and the initial shape 3 are not exactly equal. This is because another developed optimization program is used to fit the geometry rather than the area. In all cases tested the area is kept unchanged in the optimization algorithm. This means the area for the initial shape 1 and the area for the optimal shape in the tested where the initial shape 1 is the starting guess are exactly the same. So does the initial shape 2 and the initial shape 3.

The results produced by the new method are given in the following table

Type	Starting guess	Tolerance	Its	Value of F	Final shape
Without KIA	Fig 4	2.5e-3	94	-0.062323	Fig 9
With KIA	Fig 4	2.5e-3	37	-0.062333	Fig 14
Without KIA	Fig 10	4.5e-3	62	-0.062110	Fig 11
With KIA	Fig 10	4.5e-3	33	-0.062072	N/S
Without KIA	Fig 12	3.5e-3	48	-0.062280	Fig 13
With KIA	Fig 12	3.5e-3	18	-0.062302	Fig 15

Table 2: Comparison for 2-dimensional shape optimization using the knot insertion algorithm and not using the knot insertion algorithm

In the above table, KIA is an acronym for knot insertion algorithm, N/S means not shown, Its refers to iteration numbers.

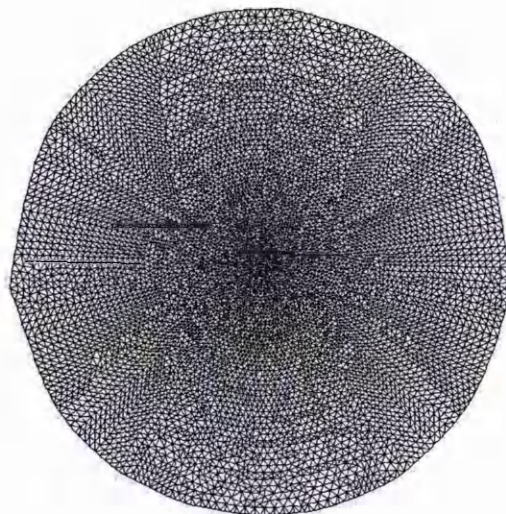


Figure 14: Optimum shape for initial shape 1 by using the new method. The knot insertion algorithm has been employed. The final number of control points are 40

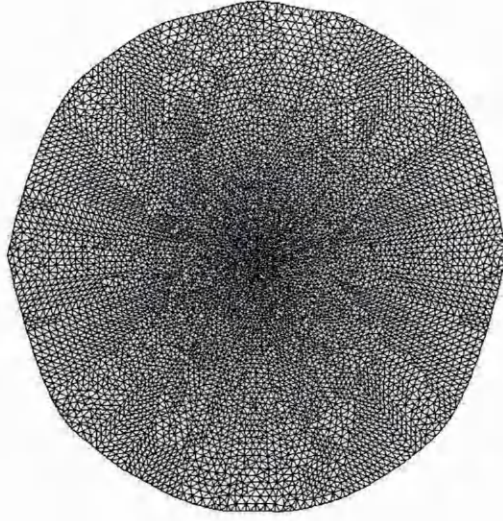


Figure 15: Optimum shape for initial shape 3 by using the new method. The knot insertion algorithm has been employed. The final number of control points are 40

For the different initial shapes as shown in the above table and pictures, the new method starts from 22 control points, and the initial mesh has about 3300 triangles. When the knot insertion algorithm is required, additional knots are uniformly inserted into the existing knot set, and the number of control points becomes 40. When the mesh refinement starts, a fine mesh is produced which has about 13,000 triangles.

The middle point in the lower B-spline curve is chosen as the special point to keep the constraint, i.e. to keep the area constant. In order to be consistent when the knot insertion algorithm starts, the special point should be selected to be the middle point in the new lower B-spline curve. If the number of control points for the new lower B-spline curve is a even number, then it will not

damage the optimization no matter which middle point (the middle point in the upper or lower B-spline curves) is selected as the special point. This has been tested.

The idea of the using the knot insertion is presented in the previous chapter but there are different ways to decide where to insert an additional knot. For example, one can locally insert knots to achieve the constraints on the curvature instead of inserting knots uniformly. It would be useful to insert quite so few knots into the geometry segment where the curvature is lower and to insert more knots into the geometry segment where the curvature is higher.

It has been shown very clearly from the above table that with the help of the new method the computational cost is decreased significantly. For the case where the starting guess is initial shape 1, the number of optimization iterations is decreased from 94 to 37 by using the knot insertion idea of the new method. For the two other cases where the starting guess is initial shape 2 and initial shape 3, the number of optimization iterations is decreased from 48 to 18 and from 62 to 33 respectively.

The criteria to start the knot insertion algorithm is set as $|F_{k+1} - F_k| < \eta$; The gaining experience in this project shows that the solution is not very sensitive to the value of η as long as the value is not too large, normally η can be selected from $\frac{\epsilon}{10}$ to $\frac{\epsilon}{2}$.

4.1.5 CPU time

The CPU time distribution could be very useful for improving the code efficiency of the developed method. It tells where the major computational cost is. Thus we can find out where to improve the code. Since the code is based on two other softwares: the mesh generator BAMG and the PDE solver CRANDNS-Poisson, the CPU time consumed on these two is taken into consideration as well.

	mesh time	solve time	time for one iteration	average time
NMF	0.4300	0.1493	35.2420	0.2433
MF	1.5996	1.1395	132.3014	0.2356

Table 3: CPU time for the optimization scheme in 2-dimensional case

Note: the explanation for this table can be found in the following table.

In the above table, NMF means no mesh refinement, MF means mesh refinement. A mesh without using mesh refinement has about 3300 elements and 1600 nodes. A mesh using mesh refinement has about 13000 elements and 7000 nodes. This table shows that the mesh time increases linearly when the number of nodes increases linearly. The reason for this is that the mesh time includes the mesh generating time and the time for mesh file Input/Output, which is normally linear on the number of nodes.

For the solver time, the above table shows that the CPU time increases quadratically when the number of nodes increase. This indicates the efficiency of the PDE solver used.

CPU time for File I/O

file description	file name	file size	CPU time
input geometry file	mesh2d1.dat	4,628B	0.000396
output mesh file	mesh2d1.msh	118,079B	0.00186
input mesh file for PDE	test.grd	198,373B	0.1493
output file from PDE solver	record1.out	38,589B	0.0239804

Table 4: CPU time for File I/O in 2-dimensional case

Note: 1. Second is chosen as the time unit, and byte is chosen as the size unit.

2. The mesh time and solve time are computed by the softwares themselves. These values are obtained from the display message. When computing the CPU time for one iteration, the display message has already been commented out.

3. Average time means the time for the optimizer to deal with one thousand triangles.

4. The mesh time includes the CPU time for its own File I/O.

Based on the data it can be found that the File I/O takes small percentage of the total computational cost after calculation. Therefore it is not necessary to change the File I/O operation in the code.

4.2 Results for a new objective function

4.2.1 Description

In this subsection the new optimization scheme is applied to a different objective function. The new objective function is chosen as:

$$F = - \int_{\Omega} \Phi dS, \quad (4.15)$$

and is subject to two constraints:

1. Area of the domain, $\Omega = k$, k is a constant.
2. The same Poisson equation as previous test case with the same boundary condition.

4.2.2 Compute the objective function

In CFD, it is almost impossible to obtain an analytical solution for the PDEs involved, even for the simple ones. Therefore the objective function must be computed discretely in most cases. Because the PDE solver uses triangular elements it is clear that:

$$\int_{\Omega} \Phi dS \approx \sum_{i=1}^{Ne} V_{abc}$$

where Ne represents the number of triangular elements, V_{abc} represents the volume of a three dimensional object that has three base vertices a, b, c and the top three vertices d, e, f are displaced by the height Φ_a, Φ_b, Φ_c respectively. (Φ_a, Φ_b, Φ_c are the values of the discrete solution of the Poisson equation at the points a, b, c respectively).

After some calculation, the following equation is obtained:

$$\int_{\Omega} \Phi dS \approx \frac{1}{3} \times \sum_{i=1}^{Ne} (S_{abc} \cdot (\Phi_a + \Phi_b + \Phi_c)), \quad (4.16)$$

where S_{abc} denotes the area of the triangle abc . The above equation is used in the code to compute the new objective function.

4.2.3 Results

For this new objective function, different initial shapes and different values of the tolerance placed on the magnitude of the gradient vector are tested. In all cases, the code converges and it finds the optimum shape. The relevant data are provided below:

Starting guess	Tolerance	Iteration numbers	Value	Final shape
Fig 4	3.0e-3	38	-0.0241948	Fig 16
Fig 4	2.0e-3	46	-0.0243138	Fig 17
Fig 4	1.5e-3	51	-0.0243496	Fig 18
Fig 12	1.5e-3	47	-0.0244045	Fig 19

Table 5: Optimization results for the new objective function in 2-dimensional case

Please note that Fig 16, Fig 17, Fig 18, and Fig 19 are included for illustrating the geometry information only. The reason that the iteration number increase rapidly with the tolerance is that the mesh used in this case is not refined enough.

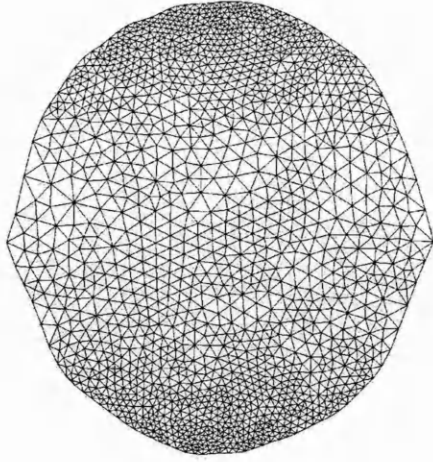


Figure 16: Optimum shape for initial shape 1 with $\|\mathbf{g}\| < 3.0e - 3$

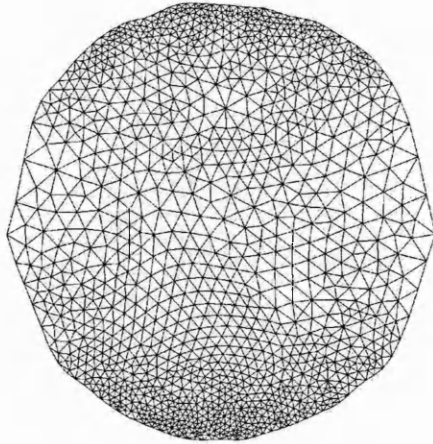


Figure 17: Optimum shape for initial shape 1 with $\|\mathbf{g}\| < 2.0e - 3$

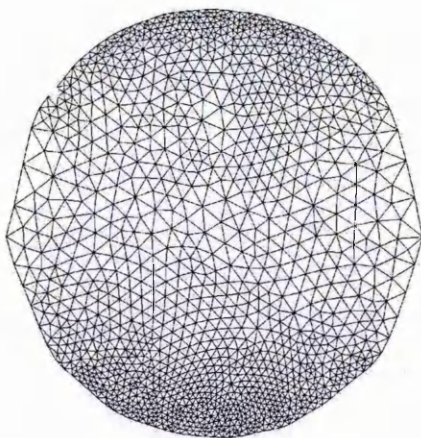


Figure 18: Optimum shape for initial shape 1 with $\|g\| < 1.5e - 3$

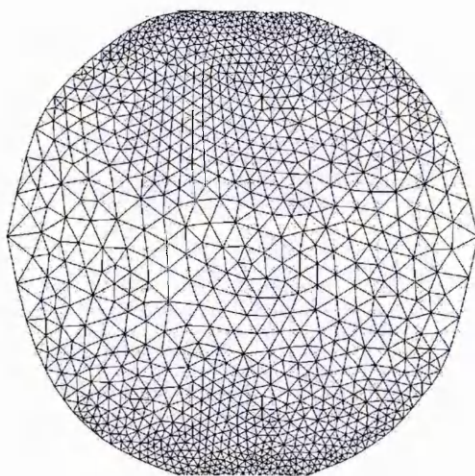


Figure 19: Optimum shape for initial shape 3 with $\|g\| < 1.5e - 3$

4.3 Comparing with BFGS

The BFGS method is regarded as one of the best methods in optimization, and hence it is chosen for comparison with the developed method in this the-

sis. The BFGS method used here has been added to the following modules in order to deal with the constrained shape optimization problems: super-reduced implementation, the geometry model, mesh refinement. The convergence history comparison between the new method and the BFGS method is shown in the figures of this section where the initial shapes and mesh refinement used are exactly the same.

The comparisons have been done on two different platforms:

1. Sun Solaris (Operating System) v880b (Super computer, 72 processors, ultraSparc 1.2G, 288G shared memory)
2. Linux chalkhill (Normal work station, CPU: P 2.4G, 1.0 G memory)

Table 6 gives the performance comparison between the new method and the BFGS method on platform 1 (v880b). In Table 6 INCP refers to the initial number of control points and FNCP refers to the final number of control points after using the knot insertion algorithm.

Methods	INCP	FNCP	Iteration number	F
BFGS	21	21	66	-0.0622820407152
the new method	12	21	37	-0.06233296118

Table 6: Performance comparison with BFGS method (without using mesh refinement)

Table 7 gives the performance comparison between the new method and the BFGS method on platform 2 (chalkhill). Both of these two methods use mesh refinements

Methods	IREF	IWREF	Iteration number	F
BFGS	16	30	46	-0.0623184405267
the new method	18	19	37	-0.0623148640617

Table 7: Performance comparison with BFGS method (using mesh refinement)

Table 8 gives the CPU time taken for these two methods:

Methods	IREF	IWREF	CPU time	CPU time(W)	Total
BFGS	16	30	6.598sec	24.902	852.628
the new method	18	19	3.524sec	24.544	529.768

Table 8: CPU time spend on BFGS method and the new method

In the Table 7 and Table 8,

- IREF refers to the numbers of iteration without mesh refinement
- IWREF refers to the numbers of iteration with mesh refinement
- CPU time (W) refers to the CPU time for one optimization iteration which uses a refine mesh
- CPU time denotes the CPU time for one optimization iteration which uses no mesh refinement (coarse mesh is always used)
- Total refers to the total CPU time for the whole optimization.
- Seconds are selected as units.

Table 8 shows that the new method saves about 38 percent of the CPU time compared to BFGS method in the test case.

The LOG curve of the convergence rate is given below:

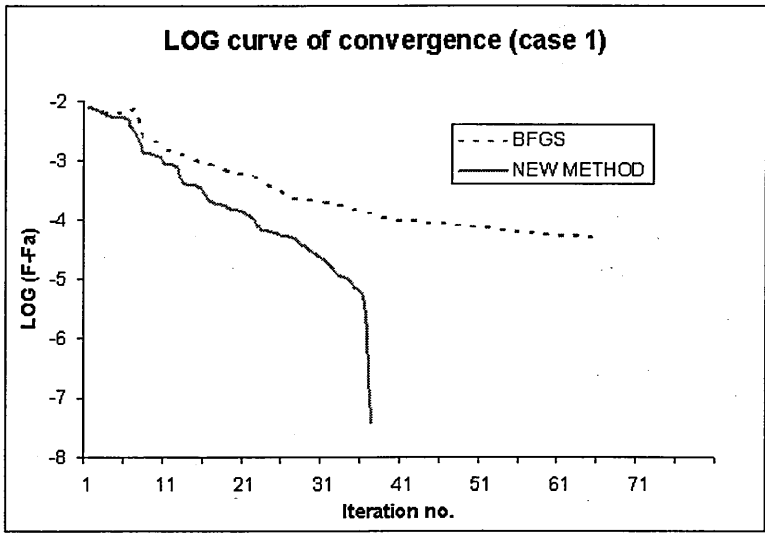


Figure 20: Performance comparison with BFGS the method (without using mesh refinement). The BFGS method used in this case does not use the mesh refinement. The purpose of doing so is to show how important mesh refinement is in geometry optimization

In Fig. 20, in iteration 7, the value of the objective function actually increases instead of decreases. This is because a potential self-intersecting shape has been detected, and the algorithm produces a new point as a replacement.

Fig. 20 shows that the BFGS method takes 66 iterations while the new method takes only 37 iterations. Furthermore, the value of the objective function is smaller than the BFGS method. This indicates how important the mesh refinement is.

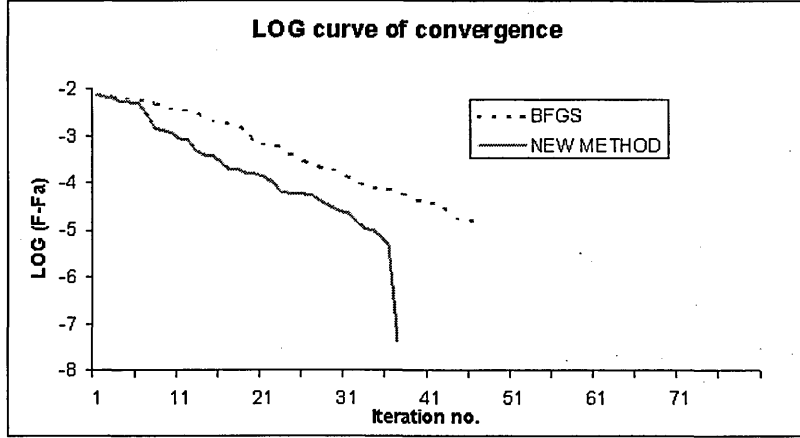


Figure 21: Performance comparison with the BFGS method (using mesh refinement)

Fig. 21 shows that the BFGS method takes 46 iterations while the new method takes only 37 iterations, saving about 20 percentages CPU time in terms of number of optimization iterations.

4.4 Error analysis

In this section, we use the following notation:

Notation

- d_i the i -th control point in the B-spline curve
- ϵ_{Δ} the triangulation error
- ϵ_{α} the approximation error caused by linear approximation of a curve which is a part of ϵ_{Δ}
- ϵ_r the residual error which is the norm of residual vector $\|AX - B\|$ in solving PDE after
- ϵ_{ls} the least square error, which is the L_2 norm of the distance between the exact point and the approximated point

In order to analyse the error, it is necessary to first define it. Hence the following term, *root mean square error*, K is defined.

$$K = \sqrt{\sum_{i=1}^n \frac{(F_i - F_i^{exact})^2}{n}} \quad (4.17)$$

$(F_i - F_i^{exact})$ is the difference between the exact solution at node i and the approximate solution at the same node.

It is easy to see that the difference between the approximate value (value of the objective function produced by the code) and the exact value is influenced by the following items:

- the least square error; because a finite number of control points are used to approximate a simple shape in which the Poisson equation has an analytical solution.
- the triangulation error; because size of the element used in the PDE solver cannot be zero, i.e, the mesh size cannot be zero.
- the residual error of the PDE solver

4.4.1 Analysis of the influence of triangulation error

In order to analyse the influence of triangulation error, the shape cannot be described by B-spline curves, otherwise the influence of least square error will automatically be included. Furthermore, the influence of residual error

must be excluded as well. This is very easy to fulfill by putting a very small tolerance on ϵ_r (it is chosen as $1e-8$ in the solver).

A circle is chosen as the input shape because for the Poisson's equation used this shape has an analytical solution. The number of boundary points is doubled and all of the boundary points are evenly distributed to see how the solution changes. Before doing this, it is best to know how the approximation error will change when the number of boundary points is doubled.

This question has a simple answer. The method to compute it analytically is as follows:

$$s_1 = \frac{1}{2}R^2 \times (\theta - \sin(\theta))$$

In this equation R is the radius of the circle, θ is the angle, $R^2 \cdot 0.5 \cdot \theta$ is the area of the segment, $\frac{1}{2}R^2 \sin(\theta)$ is the area of the triangle, s_1 is the area difference and reflects the approximation error. After the number of boundary points have been doubled, the new area difference is given by

$$s_2 = \frac{1}{2}R^2(\theta - 2 \times \sin(\frac{\theta}{2}))$$

Therefore

$$\frac{s_2}{s_1} = \frac{\frac{1}{3!}(\frac{\theta}{2})^3 + \dots}{0.5 \cdot \frac{1}{3!} \cdot \theta^3 + \dots} \quad (4.18)$$

where the neglected terms are higher order terms of Taylor series for the sin function. From this equation, it is easy to see that $\frac{s_2}{s_1} \approx 0.25$ and hence the approximation error decreases *quadratically*.

Table 9 reflects the influence of the triangulation error (not the approximation error) on the PDE solution:

Number of boundary points	Number of nodes	Error in solution	Rate
40	209	0.000123452	
80	786	2.8078e-05	4.39
40	814	0.000216005	
80	3169	5.31805e-05	4.06

Table 9: Triangulation error for different number of nodes in 2-dimensional case

From this table, it is clear that when the mesh size decreases by $1/2$, the error in the solution will decrease by $1/4$. The error analysis here is based on the real circle not on the approximated circle consisting of B-spline curves, since the influence of the least square error has to be excluded.

The conclusion for triangulation error is: if the numbers of boundary points is increased, and hence the total number of nodes is increased, (then the mesh size become a half of before) then the approximation error will decrease quadratically and therefore the error caused by triangulation error in the PDE solution also decreases quadratically. That is to say: triangulation error is a quadratic function of the mesh size.

4.4.2 Analysis of the influence of least square error

The least square error is introduced because B-spline curves are used to approximate the circle. By increasing the number of control points this error

can be decreased.

Table 10 shows the influence of the triangulation error and least square error on the PDE solution:

NCP	Number of nodes	NBP	$\epsilon_{\Delta} + \epsilon_{ls}$	Boundary error	Rate
42	200	40	0.000524489	9.82473e-05	
42	820	80	0.000199532	0.000199531	
84	810	80	0.000176791	2.68786e-05	7.27459
42	3258	80	0.00021945	0.000199531	
84	3217	80	0.00021768	2.68786e-05	7.27459

Table 10: Error (triangulation error plus least square error) for different number of nodes in 2-dimensional case

NCP: number of control points, NBP: number of boundary points, Boundary error refers to error on the boundary. Based on the boundary condition stated Section 4.1, the solution on the boundary (B-spline curves) is zero. However, according to the analytical solution, this may not be zero, it depends on how good the geometry approximation is. This is so called the boundary error.

According to the above table, if the number of control points (NCP) increases, the least square error decreases superlinearly. In fact one can see the influence of the least square error on the solution is very small. It is about 0.1 of the triangulation error as indicated by Table 10. The main error is caused by the triangulation error.

4.4.3 Conclusion on error analysis

From the above, we conclude that the main error in the code is caused by triangulation error (discretization error). The main error also has the influence of the least-square error, but this can be decreased significantly by increasing the number of control points and in fact the influence here is quite small. The triangulation error can be decreased by increasing the number of boundary points and distributing these points evenly. For the solver used, two tests have been done to verify it works correctly, see Appendix C.

4.5 The control points for the initial shapes

In order to show that the method is able to deal with different initial shapes, it is necessary to provide the value of the control points. To determine their values, another optimization program has been developed. The purpose of this program is to find a B-spline which can best fit the desired shape. If a desired initial geometry is denoted by Γ and a point on this shape is denoted as $\mathbf{P}_\Gamma$. Then the purpose is to minimize:

$$\sum \|(\mathbf{P}_\Gamma - \mathbf{P})\|$$

where $\mathbf{P}$ is the point on the B-spline which is used to fit the desired geometry and the summary extended to the all of the points in the geometry. Numerically 4000 data points have been chosen to do the fitting as stated by the above equation.

The above term can be simplified in the 2-dimensional case

$$\sum |(Y_{\Gamma} - Y)|$$

where Y_{Γ} , Y represent the y coordinate of the point on the desired curve and the B-spline curve respectively and are real numbers.

In the developed code, the objective function is chosen as

$$F = \sum (Y_{\Gamma} - Y) * (Y_{\Gamma} - Y).$$

This objective function is obviously twice differentiable with respect to variable Y while Y is dependent on the control points of the B-spline.

The gradient of the objective function with respect to the control points is computed by the central difference method

$$\mathbf{g}_i = \frac{\partial F}{\partial d_i} \approx \frac{F(d_1, d_2, \dots, d_i + \epsilon, \dots, d_n) - F(d_1, d_2, \dots, d_i - \epsilon, \dots, d_n)}{2.0 * \epsilon} \quad (4.19)$$

where $(d_1, d_2, \dots, d_i, \dots, d_n)$ are control points of the B-spline.

With this gradient vector it is easy to apply any kind of optimization algorithm. The values of the control points for different initial shapes are listed Table 11:

No	Shape 1	Shape 2	Shape 3	Shape 4	Shape 3.1	Shape 4.1
1	0	0	0	0	0	0
2	0.025867	0.0408329	0.18048	0.0290094	0.286947	0.0599536
3	0.0742828	0.111956	0.221004	0.0874254	0.34098	0.254257
4	0.153006	0.238032	0.325725	0.17432	0.470697	0.408272
5	0.223829	0.338912	0.371218	0.262156	0.498864	0.744709
6	0.308022	0.483025	0.419559	0.34842	0.498864	0.744709
7	0.366542	0.540935	0.448218	0.437529	0.470697	0.408272
8	0.484571	0.513431	0.473225	0.521352	0.34098	0.254257
9	0.656232	0.528036	0.488264	0.615202	0.286947	0.0599536
10	0.773009	0.519477	0.498164	0.68886	0	0
11	0.987758	0.525564	0.500846	0.81275		
12	0.773011	0.519477	0.498164	0.68886		
13	0.656227	0.528036	0.488264	0.615202		
14	0.484578	0.513431	0.473225	0.521352		
15	0.366566	0.540935	0.448218	0.437529		
16	0.308008	0.483025	0.419559	0.34842		
17	0.223837	0.338912	0.371218	0.262156		
18	0.153001	0.238032	0.325725	0.17432		
19	0.0742865	0.111956	0.221004	0.0874254		
20	0.0258651	0.0408329	0.18048	0.0290094		
21	0	0	0	0		

Table 11: Value of control points for different initial shapes in 2-dimensional case

In the above table, the value of the control points for these shapes has been truncated; the terminal condition of optimization for all of these different initial shapes is the same: $\|g\| < 1.0e - 8$

The mathematical expressions for these different initial shapes are listed below:

shape 1 (star shape, Fig. 12)

$$y = \begin{cases} 0.4083 * x / 0.3 & 0 \leq x < 0.3 \\ 2.3 * 0.4083 + 6.5 * 0.4083 * (x - 0.5) & 0.3 \leq x < 0.5 \\ 2.3 * 0.4083 - 6.5 * 0.4083 * (x - 0.5) & 0.5 \leq x < 0.7 \\ 0.4083 * (1.0 - x) / 0.3 & 0.7 < x \leq 1.0 \end{cases} \quad (4.20)$$

shape 2 (poly shape, Fig. 10)

$$y = \begin{cases} 4.0 * 0.5228 * x & 0 \leq x < 0.25 \\ 0.5228 & 0.25 < x \leq 0.75 \\ 4.0 * 0.5228 * (1.0 - x); & 0.75 < x \leq 1.0 \end{cases} \quad (4.21)$$

shape 3 (half circle)

$$y = \begin{cases} 2.0 * 0.78539815 * x & 0 \leq x < 0.5 \\ 2.0 * 0.78539815 * (1.0 - x) & 0.5 < x \leq 1.0 \end{cases} \quad (4.22)$$

shape 4; (nearly rectangle shape, Fig. 4)

$$y = \sqrt{0.25 - (0.5 - x)^2} \quad 0.0 \leq x \leq 1.0$$

Shapes 3.1 and 4.1 in Table 11 have the same mathematical expression as shapes 3 and 4. The only difference is that the number of controls to fit the geometry is different.

4.6 Using high order B-splines

To solve a general shape optimization problem, the user can first input control points as design variables to generate the shape, and then use a mesh generator to produce the required mesh with an input geometry file. Then the mesh file with an acceptable format will be sent to the PDE solver from which the value of objective function is obtained. The optimizer will repeat this process in each iteration until the shape is as close to the optimum shape as specified.

In the test case the shape is constructed with two B-spline curves. The knot set for each B-spline is a nominal uniform knot set. In this section the purpose is to present results for comparing the performance when using B-spline curves of different orders to see whether the high order B-splines can enhance the performance of the method.

The optimizer we have developed has a knot insertion algorithm incorporated. With help from this algorithm the optimizer will start with only a few number of control points but it will end with nearly double the number of control points or even more if such a requirement is specified.

4.6.1 Issues with the knot insertion algorithm

For a B-spline curve before the knot insertion algorithm is used the knot set is appeared as below:

$$(0, 0, \dots, 0, \underbrace{\frac{1}{n-k+1}, \frac{2}{n-k+1}, \frac{3}{n-k+1}, \dots, \frac{n-k}{n-k+1}}_{n-k}, \underbrace{1, 1, \dots, 1}_k)$$

where k represents the order and n represents the number of control points.

After the knot insertion algorithm, the new knot set becomes:

$$(0, 0, \dots, 0, \underbrace{\frac{1}{2*(n-k+1)}, \frac{2}{2*(n-k+1)}, \dots, \frac{2*(n-k)+1}{2*(n-k+1)}}_{2*(n-k)+1}, \underbrace{1, 1, \dots, 1}_k)$$

Now the optimizer will have $2 \times n - k + 1$ control points.

Two kinds of experiment have been done on high order B-spline curves. One experiment compares the results which have the same number of final control points. The other experiment compares cases which do not necessarily have the same number of final control points. (the initial number of control points is nearly the same in this case)

In the following tables in this section:

1. Iterations refers to number of optimization iterations;
2. F refers to value of the objective function;

3. EC refers to the end condition;
4. ORDER refers to the order of the B-spline curves;
5. INCP refers to the initial number of the control points in one B-spline curve;
6. FNCP refers to the final number of the control points in one B-spline curve.

Experiment 1

No	ORDER	INCP	FNCP	Iterations	F	EC
1	4	12	21	37	-0.06233296118	$\ g\ \leq 2e - 3$
2	6	13	21	41	-0.062298593697	$\ g\ \leq 2e - 3$
3	5	13	22	37	-0.0623247386731	$\ g\ \leq 2.1e - 3$
4	7	14	22	40	-0.0623337240795	$\ g\ \leq 2.1e - 3$

Table 12: Efficiency investigation for using high order B-splines (case 1)

In this experiment, it seems that if the number of final control points is fixed, then as the order becomes higher it takes more iterations to converge. This means high order will possibly decrease efficiency. On the accuracy issue, from order 4 to order 6, high order does not improve the accuracy, but in another test case, from order 5 to order 7, it appears to improve the accuracy.

Experiment 2

No	ORDER	INFCP	FNFCP	Its	F	EC	CPU (hours)
1	4	26	50	44	-0.062333	$\ g\ \leq 2.1e - 3$	1.58610
2	5	26	48	45	-0.062330	$\ g\ \leq 2.1e - 3$	1.69173
3	6	24	42	44	-0.062297	$\ g\ \leq 2.1e - 3$	1.44299
4	7	24	40	40	-0.062334	$\ g\ \leq 2.1e - 3$	1.09336

Table 13: Efficiency investigation for using high order B-splines (case 2)

In Table 13 INFCP and FNFCP refer to the initial number of free control points and the final number of free control points respectively. Table 13 shows that when the order of B-splines changes from 4 to 5, the CPU time, the number of optimization iteration, and the value of objective function become slightly worse. When the order of B-splines changes from 4 to 6, the CPU time used becomes slightly less. Moreover, when the order of B-splines changes from 4 to 7, then there is a significant saving in CPU time, saves about 31% (from 1.58610 to 1.09336). In this experiment, the high order B-spline curve is used and the number of control points becomes smaller when the order becomes higher. In this experiment, it seems *if higher order is used and provided the number of control points decreases when the order is higher*, then the optimiser will be slightly more efficient in general. The accuracy can be also improved. It also clearly shows that when the order changes from 4 to 7. the CPU time decreases from 1.58610 to 1.09336.

4.7 Application to airfoil design

The purpose of this application is to find optimum shapes that minimize the drag coefficient (C_d) of transonic airfoils subject to some constraints on the design conditions. The constraints are:

- to keep the lift coefficient (C_l) constant and;
- to keep the maximum thickness constant.

4.7.1 The geometry representation

This application follows Zhong and Qiao's work [63]. In their work, the shape of the airfoil is formed by a base shape plus a shape change where the shape change is parameterized by a sum of shape functions coefficients with corresponding shape functions. Therefore, the shape of the airfoil is defined by the following equations.

$$Y(\Gamma^L) = Y(\Gamma_{ini}) + \sum_{i=1}^m b_i g_i(X) \quad (4.23)$$

$$Y(\Gamma^U) = Y(\Gamma_{ini}) + \sum_{i=1}^n a_i f_i(X) \quad (4.24)$$

where $Y(\Gamma^U)$, $Y(\Gamma^L)$ represent the y coordinates of the upper geometry (Γ^U) and the lower geometry (Γ^L) of the airfoil respectively, $Y(\Gamma_{ini})$ represents the y coordinate of the base, or initial airfoil, and $f_i(X)$, $g_i(X)$ are shape functions.

The shape functions are chosen based on Hicks and Vanderplaats's work [25], and are given by the following equations:

$$f_1(X) = \frac{X^{n_1}(1-X)}{e^{m_1 \times X}}, \quad (4.25)$$

$$f_i(X) = \sin^2(\pi X^{n_i}) \quad i = 2, 3, 4, 5. \quad (4.26)$$

based on the design case of RAE2822 where airfoil RAE2822 is chosen as the initial airfoil. In the above equations, $m_1 = 20$, $n_1 = 0.25$, and the maximum value occurs at about 1.2% chord position. For $i = 2, 3, 4, 5$, $n_i =$

$\log 0.5 / \log X_i$, and is used to place the peak at X_i , which occurs at 0.2, 0.4, 0.6, 0.8 respectively. In the design case based on NACA 0012 where airfoil NACA 0012 is chosen as the initial airfoil, 8 shape functions are chosen to represent the shape change, which is described in more detail later.

For the new method, the shape change is parameterized by cubic B-splines with nominal uniform knot set. The geometry of the airfoil is given by the following equations:

$$Y(\Gamma^U) = Y(\Gamma_{ini}) + \sum_{i=1}^n d_i^U N_{i,4}(u) \quad (4.27)$$

$$Y(\Gamma^L) = Y(\Gamma_{ini}) + \sum_{i=1}^m d_i^L N_{i,4}(u) \quad (4.28)$$

where d_i^U, d_i^L are the control points of the upper and lower B-splines and $N_{i,4}(X)$ are the base function of the cubic B-spline. In this 2-dimensional design case; d_i^U, d_i^L can be regarded as the value of the y co-ordinates of the control points and the x co-ordinates of the points represent the parameters, i.e, $X = u$.

4.7.2 The flow solver and the optimization algorithm

The transonic airfoil analysis code used in this paper is the NPUFOIL program developed by Qiao [48]. This program is based on a combination of a full potential equation solver with boundary layer flow analysis. The flow and the aerodynamic characteristics are computed via viscous/inviscid iteration.

In Zhong's paper [63], the direct search EXTREM, optimization method developed by Jacob [33] is used, and the new optimization algorithm developed in Chapter 3 also employed to do a comparison. The results obtained by these two optimization algorithms are compared in two different design cases. One design case is based on using RAE 2822 as the initial airfoil, the other uses NACA 0012.

4.7.3 RAE 2822

In this design case, the initial airfoil RAE 2822 is chosen. The purpose is to get an optimum shape under the design conditions: Mach number $M_a = 0.75$, and Reynolds number $Re = 2.0 \times 10^6$, and the lift coefficient C_l is set at 0.7.

By using the EXTREM method, the optimum solution of the shape coefficients stated in equation 4.24 is given as follows:

$$\begin{aligned} a_1 &= 0.39521 & a_2 &= -0.26958 \\ a_3 &= -0.030086 & a_4 &= 0.23385 \\ a_5 &= 0.20672 \end{aligned}$$

For the new method, the optimum solution of the control points is:

$$\begin{aligned} d_1^U &= -0.29105 & d_2^U &= -0.42379 & d_3^U &= -0.48731 \\ d_4^U &= -0.36079 & d_5^U &= -0.32293 & d_6^U &= 0.36791 \\ d_7^U &= 0.30644 & d_8^U &= 0.091182 & d_9^U &= 0.081497 \end{aligned}$$

The performance comparison for the two methods is given in the following table:

Optimization methods	C_d	Initial C_d	NFC	CPUPFC
EXTREM	0.0087	0.0116	599	0.22246
New Method	0.0077	0.0116	177	0.22418

Table 14: Performance comparison for the two methods in the design case of RAE 2822

In the above table NFC refers to number of function calls, CPUPFC means the CPU time per function call.

The above table shows that by using the EXTREM method, the drag coefficient is reduced by 25%, whereas using the new method the drag coefficient is reduced by about 34%. This is because that the objective function is sensitive to the geometry change and the new method works better than the EXTREM method in the tested case. Moreover, the new method uses only 177 function calls while the EXTREM method requires 599 function calls. The total CPU time used is reduced by around 70%

The comparison of the optimum shape and pressure distribution are shown in Fig. 22 and Fig. 23

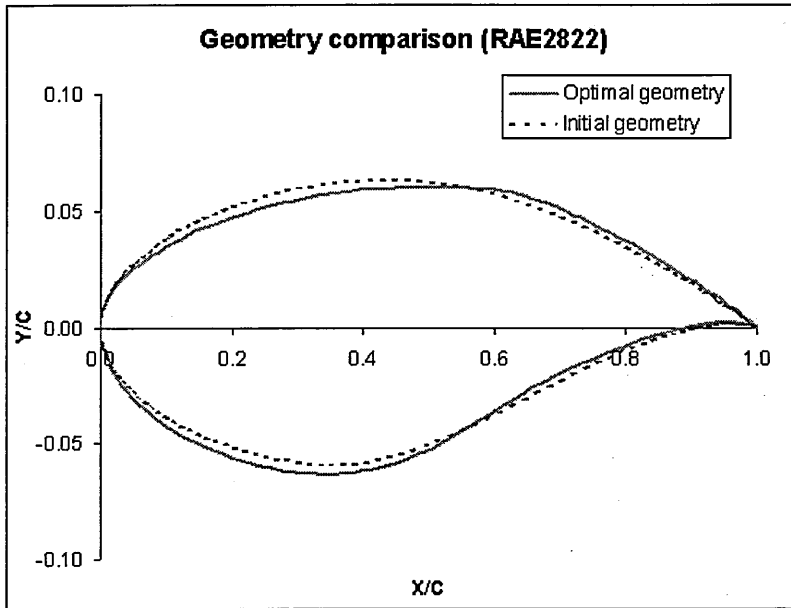


Figure 22: Initial and optimal geometry for RAE 2822. The picture shows that the geometry constraint is kept unchanged

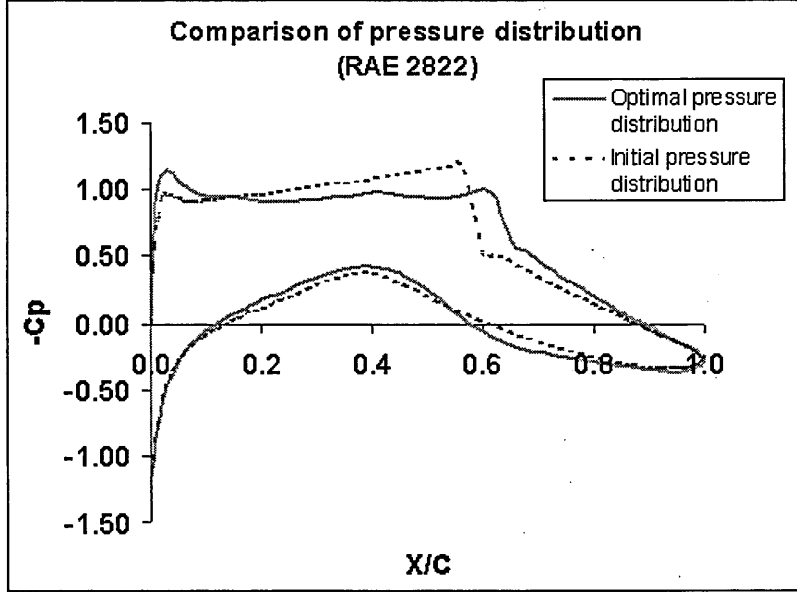


Figure 23: Initial and optimal pressure distribution for RAE 2822, The picture shows that the shock wave has been decreased

Fig. 23 shows that the strength of the shock is very much weakened which results in the reduction of the shock wave drag, hence the total drag is reduced.

4.7.4 NACA 0012

In this design case, the symmetric airfoil NACA 0012 is chosen as the initial geometry. The lift coefficient $C_l = 0.5$, Mach number $M_a = 0.77$, and the Reynolds number $Re = 6.5 \times 10^6$ are kept unchanged. In the EXTREM method, 8 shape functions are used to parameterize the geometry. In the new optimization algorithm, 5 free control points are used to parameterize the geometry and this increases to 9 free control points after the knot insertion algorithm to describe the desired geometry at a late stage.

The shape functions chosen are given by the following equations

$$f_i(X) = \sin^4(\pi X^{n_i}) \quad i = 1, 2, 3, 4, 5 \quad (4.29)$$

$$f_i(X) = \sin^4(\pi(1 - X)^{n_i}) \quad i = 6, 7, 8 \quad (4.30)$$

In the above equations, for $i = 1, 2, 3, 4, 5$, $n_i = \log 0.5 / \log X_i$ is used to place the peak at X_i , while the n_i are 0.06, 0.13, 0.2, 0.4, 0.6 respectively. For $i = 6, 7, 8$, assigning $n_6 = n_3, n_7 = n_2, n_8 = n_1$ makes the peaks occur at 0.8, 0.87, 0.94 respectively.

By using the EXTREM method, the optimum solution of the shape coefficients is:

$a_1 = -0.045544$	$a_2 = -0.21081$	$a_3 = -0.10340$
$a_4 = 0.16878$	$a_5 = 0.42756$	$a_6 = 1.6056$
$a_7 = 0.74775$	$a_8 = 0.79883$	

For the new method, the drag is reduced from 0.0182 to 0.0070. and the optimum solution of the control point is:

$d_1^U = -0.571028$	$d_2^U = -0.70975$	$d_3^U = -0.41511$
$d_4^U = 0.13373$	$d_5^U = 0.78084$	$d_6^U = 1.3277$
$d_7^U = 0.98959$	$d_8^U = 0.56642$	$d_9^U = 0.13606$

The performance comparison for the two methods is given by the following table:

Optimization methods	C_d	Iteration numbers	NFC	CPUPFC
EXTREM	0.0070	N/A	408	0.22246
New Method	0.0070	20	233	0.22418

Table 15: Performance comparison for the two methods in the design case of NACA 0012

This table shows that by using the EXTREM method and the new method the drag coefficient is reduced by about 62%. However, using the new method only requires 233 function calls which saves about 43% total CPU time.

The comparison of pressure distribution and the optimum shape are shown in the figures below:

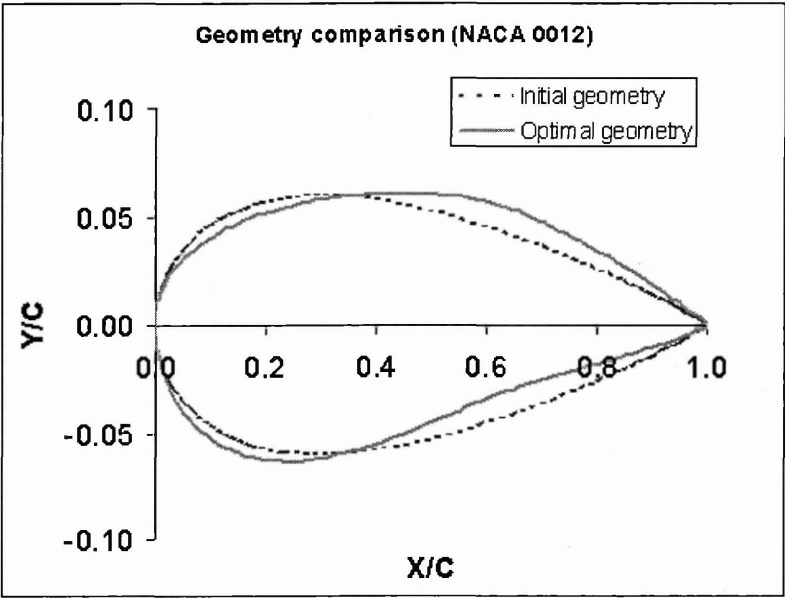


Figure 24: Initial and optimal geometry for NACA 0012. The picture shows that the geometry constraint is kept unchanged

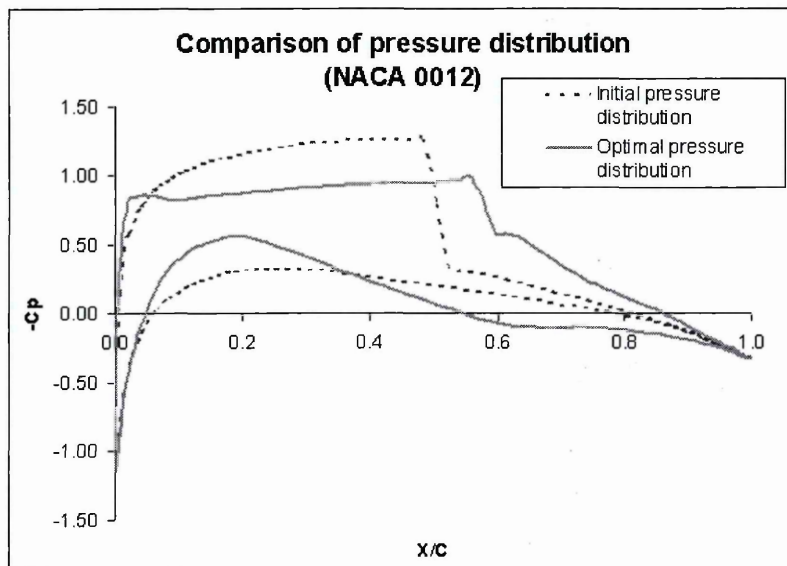


Figure 25: Initial and optimal pressure distribution for NACA 0012. The picture shows that the shock wave has been decreased

This figure shows that the strength of the shock is very much weakened which results in the reduction of the total drag.

4.8 Performance comparison for methods in the design case based on NACA 0012

In this section, the purpose is to do a performance comparison among the EXTREM method, BFGS method, and the new method. In the design case stated in this section, the initial geometry is chosen as the symmetric airfoil NACA 0012. The lift coefficient $C_l = 0.5$, $M = 0.77$, and the Reynolds number $Re = 6.5 \times 10^6$ are kept unchanged. The EXTREM method uses 5 shape function to parameterize the geometry while the new optimization algorithm uses 5 free control points to parameterize the geometry while it changes to be 9 free control points to describe the desired geometry.

By using the EXTREM method, the optimum solution is:

$$\begin{aligned}\alpha_1 &= 0.050156 & \alpha_2 &= -0.37299 \\ \alpha_3 &= -0.15513 & \alpha_4 &= 1.6377 \\ \alpha_5 &= -0.52993\end{aligned}$$

which are the coefficients for the 5 shape functions respectively.

By using the BFGS method, the optimum solution for the coefficients for the 5 shape functions stated in the previous content is:

$$\begin{aligned}\alpha_1 &= 0.42912 & \alpha_2 &= -0.91874 \\ \alpha_3 &= 0.38862 & \alpha_4 &= 0.77727 \\ \alpha_5 &= -0.45295\end{aligned}$$

For the new method, the geometry is first constructed using 5 free control points plus the base shape. Finally it reaches the optimum solution which has 9 free control points with the drag is reduced from $1.82e-02$ to $6.8e-03$. The optimum solution is

$$\begin{array}{lll} d_1^U = -0.61276 & d_2^U = -0.53544 & d_3^U = -0.49769 \\ d_4^U = 0.22370 & d_5^U = 0.89380 & d_6^U = 1.4377 \\ d_7^U = 1.0997 & d_8^U = 0.67126 & d_9^U = 0.11616 \end{array}$$

The performance of comparison for the two methods is given below:

Optimization methods	C_d	iteration numbers	NFC	CPU
EXTREM	0.0085	N/S	535	0.22246
BFGS	0.0087	21	187	0.22418
New Method	0.0068	20	181	0.22418

Table 16: Performance comparison for the three methods

In the above table, NFC means number of function calls.

The comparison with the BFGS method is given by the following graph and it clearly shows that the new method has a better convergence rate.

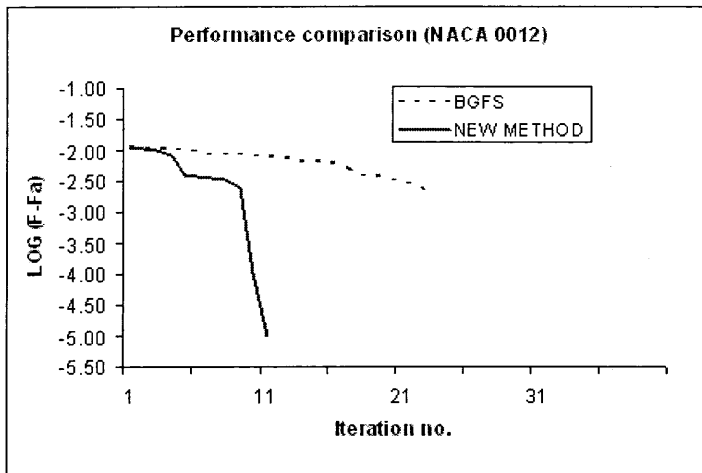


Figure 26: Comparison with the BFGS method in the NACA 0012 design case

4.9 Multiobjective Optimization Design of Transonic Airfoils

Developing new transonic airfoils which have lower drag coefficients is very important for airplane designers as it can result in significant reduction in the fuel consumption. Consequently, the direct cost for the operation of the airplane can be reduced and the airplane flight range can be increased.

Single objective optimization techniques have been extensively used in airfoil design in the past, examples can be found in [26],[25]. Although single objective optimization methods can generally find an optimum airfoil for a

specific design condition within a number of optimization iterations (which may depend on the performance of the selected optimization method) it has no inherent mechanism to handle multiple design-point problems.

Therefore, it would be critically important to use multiobjective optimization methods which would make the designed airfoil have a low drag coefficient at multiple design points. Among the multiobjective optimization methods, global methods, such as the methods based on genetic algorithms, are capable of finding the global optimum, however they are generally less efficient than the local methods. The weighted method, where the final objective function is a linear combination of specified weighted coefficients times corresponding objective functions, may be the simplest, and the most straightforward, however, there is no guarantee that the weighted method can find an optimum in a general case, see [34]. A hierarchical multiobjective optimization method where the previous optimum value of the objective function works as a constraint for the next objective function is a possible choice. By using a hierarchical approach, a multiobjective optimization problem with m objective functions will be converted into m single objective optimization problems with extra constraints.

The efficiency of a hierarchical multiobjective optimization method is largely dependent on the efficiency of the selected single objective optimization method. In this section, a multiobjective optimization method based on a hierarchical approach incorporated with a novel shape optimization method (proposed in Section 3.1) using the knot insertion algorithm has been pre-

sented and shown to save the computational cost significantly. Similar work has been done by Zhong and Qiao in [63]. In their work, the shape change is parameterized by shape functions where in this section the shape change is parameterized by B-spline shape functions, which are standard for industry and also hold some very useful properties. Furthermore, as stated in the previous content of this chapter, the novel shape optimization method has proved that it is better than the EXTREM method, one of the direct search methods, and the BFGS method in the test numerical cases.

4.9.1 The multiobjective optimization method

The main idea of this multiobjective optimization method is as follows; For a multiobjective optimization problem, stated below:

$$\min F_i(\alpha_1, \alpha_2, \dots, \alpha_n) \quad i = 1, 2, \dots, m$$

where $(\alpha_1, \alpha_2, \dots, \alpha_n)$ (denoted as a vector α) are design variables, will be converted into the following problem:

$$\min F_i(\alpha_1, \alpha_2, \dots, \alpha_n) \quad i = 1, 2, \dots, m$$

From $i=2$, to m subject to

$$F_{j-1}(\alpha) \leq (1 + \epsilon_{j-1})F_{j-1}(\alpha^{i-1}) \quad j = 2, 3, \dots, i$$

where ϵ_{j-1} is a small value acting as a relaxation parameter, and α^{i-1} is the previous optimum point, a known vector. The procedure will not terminate

until the final optimum point α^m is obtained.

By using this hierarchical approach, the above multiple objective optimization problem will be converted into m single objective optimization problems with extra constraints. These can be solved by the method presented Chapter 3 which employs the knot insertion algorithm and the super-reduced idea.

If there are only two objectives, stated below:

$$\min F_1(\alpha_1, \alpha_2, \dots, \alpha_n)$$

$$\min F_2(\alpha_1, \alpha_2, \dots, \alpha_n)$$

then the procedure is to first solve the single optimization problem:

$$\min F_1(\alpha_1, \alpha_2, \dots, \alpha_n)$$

The optimal value after the single objective optimization can be denoted as F_1^* and the optimal point can be denoted α^* . Then another constrained optimization problem must be solved to achieve the optimal solution for the multiobjective optimization problem.

$$\min F_2(\alpha_1, \alpha_2, \dots, \alpha_n) \quad \text{subject to} \quad F_1 < (1 + \epsilon)(F_1^*)$$

Obviously in the second stage the initial geometry should be the optimal geometry obtained from the single objective optimization.

4.9.2 Results for multiobjective optimization problems

In the design case based on NACA 0012, the purpose is to minimize the drag coefficients at two design Mach numbers ($Ma = 0.77$ and $Ma = 0.79$). When the single objective optimization method is used the drag is reduced by about 62% at $Ma = 0.77$ (from 0.0182 to 0.0070). However the drag at Mach number 0.79 is still relatively large. When using the multiobjective optimization method proposed above, with a 10% release on the constraint of the drag at $Ma = 0.77$, the drag at Mach number 0.79 has a further 9% decrease. Furthermore, the drag at all Mach numbers where $Ma \geq 0.79$ have been decreased, see Fig. 27. The comparison of the optimized pressure distribution with the original is shown in Fig. 28 The comparison of the geometry is shown in Fig. 29.

In the design case based on RAE 2822, we would like to have minimum drag coefficients at two design Mach numbers ($Ma = 0.77$ and $Ma = 0.79$). When the single objective optimization method is used, the drag is reduced by about 41% at $Ma = 0.77$ (from 0.0150 to 0.0088). When the multi-objective optimization method is used, the drag at $Ma = 0.79$ will have a further reduction about 26% (from 0.0182 to 0.0134). The designed airfoil has a wider; lower drag range, see Fig. 30. The comparison of the pressure distribution and the geometry are shown by Fig.31 and Fig.32 respectively.

The value for the control points for these two cases are also given below:

NACA 0012

RAE2822

0.0494235588307	0.126086421364
0.0655581555744	-0.310571240471
-0.0639794367615	-0.441785104109
-0.114741267656	-0.358605270191
-0.138489938195	-0.266374705062
-0.204873032942	-0.342032711209
0.0490139211688	0.128934354044
-0.0546455206997	-0.214546453234
0.033069471617	0.104806215404

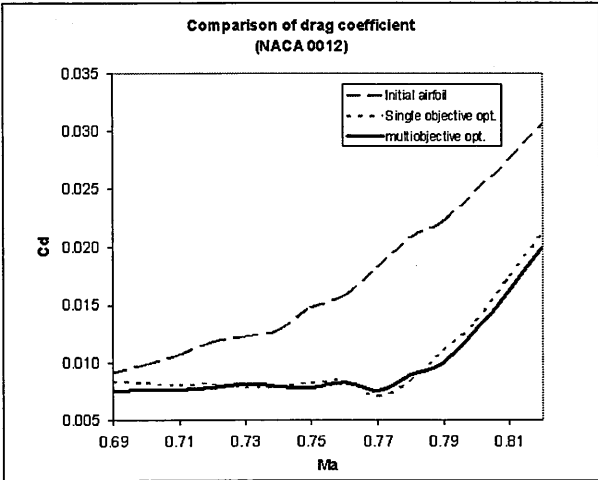


Figure 27: The comparison of drag coefficients for initial airfoil and optimal airfoil (NACA 0012)

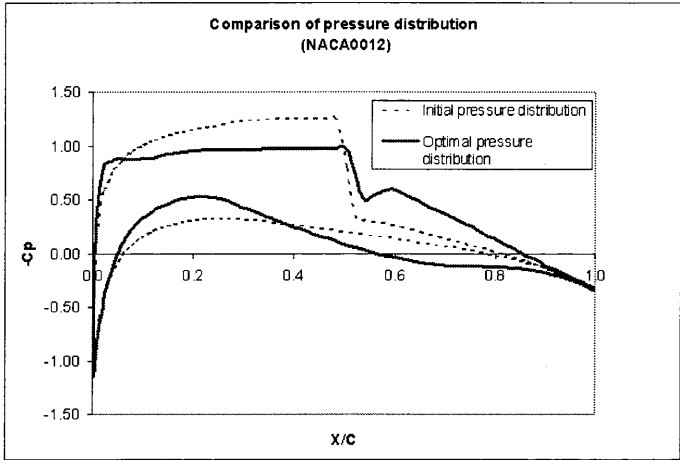


Figure 28: The comparison of pressure distribution for initial airfoil and optimal airfoil (NACA 0012)

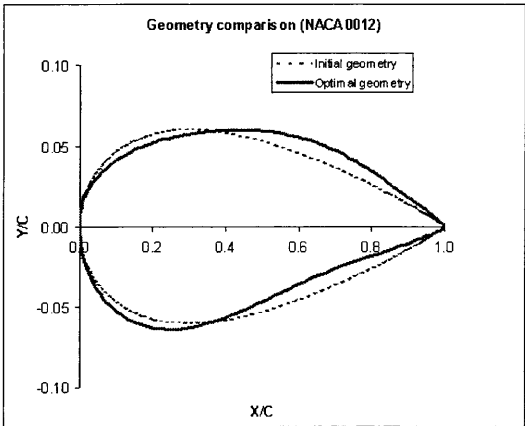


Figure 29: The comparison of geometries for initial airfoil and optimal airfoil (NACA 0012)

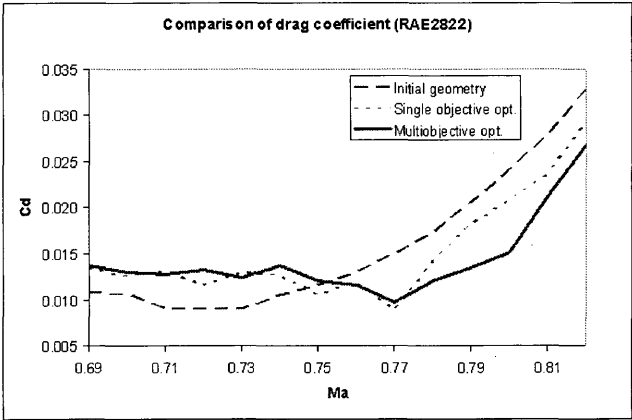


Figure 30: The comparison of drag coefficients for initial airfoil and optimal airfoil (RAE 2822)

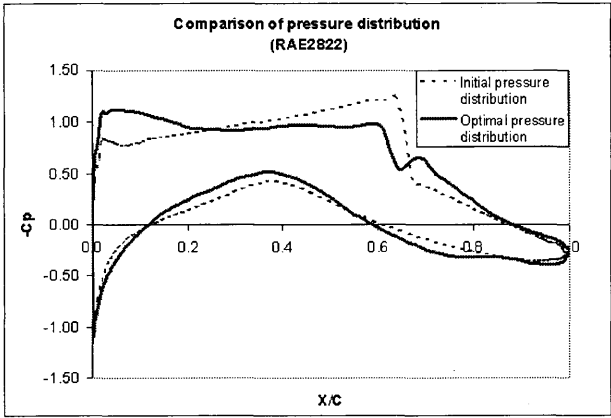


Figure 31: The comparison of pressure distribution for initial airfoil and optimal airfoil (RAE 2822)

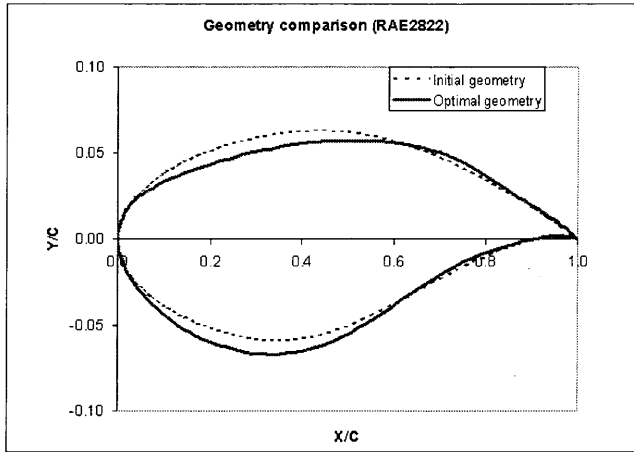


Figure 32: The comparison of geometries for initial airfoil and optimal airfoil (RAE 2822)

In conclusion, a new multiobjective optimization method in conjunction with a novel shape optimization method has been presented in this section. Application to two airfoil design cases show that the newly developed multiobjective optimization approach can not only save computational time, but also improve the aerodynamic characteristics of the airfoils to be optimized. The numerical tests show that the resultant multiobjective optimized airfoils have lower drag coefficients over a wider range of Mach numbers.

4.10 A new analysis for Poisson's problem

For the Poisson equation stated below

$$\nabla^2 \Phi = -1.0 \quad \text{in } \Omega \quad (4.31)$$

with the boundary condition

$$\Phi = 0 \quad \text{on} \quad \partial\Omega$$

The purpose is to find a shape to maximise the maximum value in the domain (Ω) :

$$\max_{\partial\Omega} \max_{\Omega} F \quad (4.32)$$

subject to a geometric constraint, here:

$$\text{Area of the domain, } \Omega = k, \quad k \text{ is a constant} \quad (4.33)$$

After reorganizing, the objective function is chosen as:

$$F = - \max_{\Omega} \Phi \quad (4.34)$$

and is subject to two constraints:

1. Area of the domain, $\Omega = k$, k is a constant.
2. The Poisson equation with the specified boundary condition.

4.10.1 A new theoretical results in shape representation

In this problem, B-splines are chosen to parameterize the geometry because: Two B-spline curves are chosen to parameterize the upper geometry and the lower geometry. In Section 4.1 of the report we made the following assumption:

the length between the first control point and the last point is fixed.

Now we remove this assumption, therefore there is **no** assumption made for this problem in order to secure generality.

Mathematical expression of B-spline

The expression for each of the B-splines curves is given by the equation below:

$$y(u) = \sum_{i=1}^n d_i N_{i,k}(u) \quad 0 \leq u \leq l \quad (4.35)$$

where l represents the length and the B-spline basis function is defined as:

$$\begin{aligned} N_{i,1}(u) &= 1 & \text{if} & & u_i \leq u < u_{i+1} \\ N_{i,1}(u) &= 0 & \text{otherwise} \end{aligned}$$

$$N_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} N_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} N_{i+1,k-1}(u) \quad (4.36)$$

The knot set $u_1, u_2, \dots, u_{n+k}$ and the order k are not specified yet.

A knot set is chosen as below:

$$\underbrace{(0, 0, \dots, 0)}_k, \underbrace{\frac{l}{n-k+1}, \frac{2*l}{n-k+1}, \frac{3*l}{n-k+1}, \dots, \frac{(n-k)*l}{n-k+1}}_{n-k}, \underbrace{l, l, \dots, l}_k$$

4.10.2 Computation of the the area with variable interval length

Based on the above content, the area of the shape is given by the equation below:

$$\begin{aligned} A(\Gamma) &= A(\Gamma^U) - A(\Gamma^L) \\ A(\Gamma^U) &= \int_0^l (y^U(x))dx \\ A(\Gamma^L) &= \int_0^l (y^L(x))dx \end{aligned} \quad (4.37)$$

provided that the shape consists of the upper geometry Γ^U and the lower geometry Γ^L are not crossed. In the above equation array, y^U, y^L denote the upper boundary and lower boundary respectively, A^U, A^L denote the area of the upper part and the area of the lower part respectively, and y^U, y^L are defined by the following equations:

$$y^U = \sum_{i=1}^{nu} d_i^U N_{i,k}(x) \quad y^L = \sum_{i=1}^{nl} d_i^L N_{i,k}(x) \quad 0 \leq x \leq l.$$

Based on the equation of computing the area, the derivative of the area with respect to the control points in the upper curve is given by the equation below:

$$\frac{\partial A}{\partial d_i^U} = \int_0^l N_{i,k}(x)dx \quad (4.38)$$

where

$$\int_0^l N_{i,k}(x)dx = \frac{u_{i+k} - u_i}{k}. \quad (4.39)$$

with $\frac{\partial A}{\partial d_{nu}^U} = \frac{l-u_{nu}}{k}$. (nu refers to the number of control points in the upper curve) Hence if a cubic B-spline curve with nominal uniform knot set is used, the gradient of the area with respect to the control points in the upper B-spline curve is given by

$$\frac{\partial A}{\partial d_i^U} = \frac{u_{i+4} - u_i}{4} \quad (4.40)$$

For the control points in the lower B-spline curve, the gradient is given by:

$$\frac{\partial A}{\partial d_i^L} = -\frac{u_{i+4} - u_i}{4} \quad (4.41)$$

with $\frac{\partial A}{\partial d_{nl}^L} = -\frac{l-u_{nl}}{4}$. (nl refers to the number of control points in the lower curve). The area is given by the following equation:

$$A = \sum_{i=1}^{nu} (d_i^U \frac{u_{i+4} - u_i}{4}) - \sum_{j=1}^{nl} (d_j^L \frac{u_{j+4} - u_j}{4}) \quad (4.42)$$

where nu and nl are the number of control points in the upper and lower curves respectively.

Now there is only one problem left, that is: $\frac{\partial A}{\partial l}$ Let us recall the knot set,

$$\underbrace{(0, 0, \dots, 0)}_k, \underbrace{\frac{l}{n-k+1}, \frac{2*l}{n-k+1}, \frac{3*l}{n-k+1}, \dots, \frac{(n-k)*l}{n-k+1}}_{n-k}, \underbrace{l, l, \dots, l}_k$$

which is a nominal uniform knot set. Then from the equation 4.42 and the knot set, we obtain the following:

$$\begin{aligned}
\frac{\partial A}{\partial l} = & (d_1^U \frac{1}{4(n-k+1)}) + (d_2^U \frac{2}{4(n-k+1)}) + (d_3^U \frac{3}{4(n-k+1)}) \\
& + (d_4^U \frac{4}{4(n-k+1)}) + (d_5^U \frac{4}{4(n-k+1)}) + \dots + (d_{nu-3}^U \frac{4}{4(n-k+1)}) \\
& + (d_{nu-2}^U \frac{3}{4(n-k+1)}) + (d_{nu-1}^U \frac{2}{4(n-k+1)}) + (d_{nu}^U \frac{1}{4(n-k+1)}) \\
& - (d_1^L \frac{1}{4(n-k+1)}) - (d_2^L \frac{2}{4(n-k+1)}) - (d_3^L \frac{3}{4(n-k+1)}) \\
& - (d_4^L \frac{4}{4(n-k+1)}) - (d_5^L \frac{4}{4(n-k+1)}) - \dots - (d_{nl-3}^L \frac{4}{4(n-k+1)}) \\
& - (d_{nl-2}^L \frac{3}{4(n-k+1)}) - (d_{nl-1}^L \frac{2}{4(n-k+1)}) - (d_{nl}^L \frac{1}{4(n-k+1)})
\end{aligned}$$

because

$$\begin{aligned}
\frac{u_{i+4} - u_i}{4} &= \frac{i * l}{4(n-k+1)} & i < k \\
\frac{u_{i+4} - u_i}{4} &= \frac{4 * l}{4(n-k+1)} & k \leq i \leq n-k+1 \\
\frac{u_{i+4} - u_i}{4} &= \frac{(n-i+1) * l}{4(n-k+1)} & i > n-k+1
\end{aligned}$$

where n, k refers to the number and the order of the B-splines.

4.10.3 Compute the super-reduced gradient by using the finite difference method

Since the objective function F is a function of the state variable Φ and the shape $\partial\Omega$ ($F(\Phi, \partial\Omega)$), and the relationship between Φ and $\partial\Omega$ is determined

by the Poisson equation with specified boundary condition, the state equation (the Poisson equation) is solved first in each optimization iteration.

$$F(\Phi, \partial\Omega) \longrightarrow F(\partial\Omega)$$

Second, because $\partial\Omega$ is a function of design variables, i.e., control points of the B-spline curves, $(\partial\Omega(d_1, d_2, \dots, d_n), l, d_1, d_2, \dots, d_n, l$ are the control points of the B-spline curves), and the area of the domain needs to be fixed, one of the control points is moved in order to keep the area constant. In this case, the constraint equation $A = k$ needs to be solved as well.

Now the problem is redefined as:

$$F = F(d_1, d_2, \dots, d_{i-1}, d_{i+1}, \dots, d_n, l) \quad (4.43)$$

with d_i eliminated in each optimization iteration.

The justification for this method is given in the Appendix B. Based on the proof, if d_i can be uniquely defined by $(d_1, d_2, \dots, d_{i-1}, d_{i+1}, \dots, d_n), l$, then the optimal point produced by the super-reduced idea is also the optimal point for the Langrage system.

The super-reduced gradient is computed by using the following finite dif-

ference method so that the constraints are kept unchanged:

$$\frac{\partial F}{\partial l} \approx \frac{F(d_1, d_2, \dots, d_i + \delta, \dots, d_n, l + \epsilon) - F(d_1, d_2, d_3, \dots, d_n, l)}{\epsilon} \quad (4.44)$$

$$\frac{\partial F}{\partial d_j} \approx \frac{F(d_1, d_2, \dots, d_j + \epsilon, \dots, d_i + \delta, \dots, d_n, l) - F(d_1, d_2, d_3, \dots, d_n, l)}{\epsilon} \quad j \neq i \quad (4.45)$$

where $j = 1, 2, \dots, n$, $j \neq i$ (if $j = i$, then $\frac{\partial F}{\partial d_i} = 0$) and δ is given by solving the constraints:

$$\text{Area of } \Omega (y_1, y_2, \dots, y_{i-1}, y_i + \epsilon, y_{i+1}, \dots, y_n + \delta, l) = \text{constant.}$$

4.11 Summary

A new efficient method for shape optimization is given. It has been shown that this method has good performance in both a number of mathematical problems and industrial applications. Using this method, researchers will be able to accelerate their own optimization schemes as long as they can use B-spline to parameterize the geometry.

In the problem where the Poisson equation is the governing equation, the method finds the optimum shapes for varying initial guesses, differing objective functions and differing numbers of control points. With the new method,

the CPU time has been decreased. We have shown that the method is more efficient than the BFGS method, one of the best known methods in optimization in several test cases.

In airfoil design, the drag has been reduced significantly and much more efficiently compared with other methods. In the design case based on RAE 2822, using the EXTREM method, the drag coefficient has been reduced by 25%. (from 0.0116 to 0.0087) with 599 function calls. Using the new method, the drag coefficient has been decreased by about 34% percentages (from 0.0116 to 0.0077) with only 177 function calls giving a saving of CPU time of about 70%. In the design case of NACA 0012, by using the EXTREM method the drag coefficient has been reduced by about 62%. (from 0.0182 to 0.0070) with 408 function calls while using the new method, the drag coefficient has been decreased by about 62% (from 0.0182 to 0.0070) with only 233 function calls giving a saving of CPU time about 43%.

The use of high order B-spline curves has been investigated although not exhaustively. From the experimental results, we see that if the number of control points is fixed, then using the high order B-spline curve does not necessarily improve the efficiency or the accuracy. On the other hand, if a high order B-spline with a relatively small number of control points is used, then it will generally improve both the efficiency and the accuracy of the optimiser.

A multiobjective optimization method in conjunction with a novel shape optimization method for multiple design-point problem has also been devel-

oped and applied to two design cases. In the design case based on NACA 0012, if the single objective optimization method is used, the drag is decreased by about 62% at $Ma = 0.77$ (from 0.0182 to 0.0070). When using the multiobjective optimization method with a 10% relaxation on the constraint of the drag at $Ma = 0.77$, the drag at Mach number 0.79 will have a further 9% decrease. In the design case based on RAE2822, if the single objective optimization method is used, the drag is decreased by about 41% at $Ma = 0.77$ (from 0.0150 to 0.0088). When the multiobjective optimization method is used, the drag at $Ma = 0.79$ will have a further deduction about 26% (from 0.0182 to 0.0134). The designed airfoil appears to have a wider lower drag range.

5 Numerical experiments in 3-dimensional case

In this chapter, results for the 3-dimensional shape optimization problem using the new method are reported. The example is a PDE-constrained optimization problem with an equality geometry constraint. The Poisson equation is chosen as the governing equation because solving this equation is simpler than other non-linear PDEs. Furthermore, the solution procedure of shape optimization problems is basically the same, independent of the governing equation. The data and results are given.

5.1 Application for using Poisson's equation as the governing equation

For the 3-dimensional Poisson equation stated below

$$\nabla^2 \Phi = -1.0 \quad \text{in } \Omega \quad (5.1)$$

with the boundary condition

$$\Phi = 0 \quad \text{on } \partial\Omega$$

The purpose is to find an optimum shape to maximise the maximum value in the domain (Ω) :

$$\max_{\partial\Omega} \max_{\Omega} \Phi \quad (5.2)$$

subject to a geometric constraint here;

$$\text{Volume of the domain } (\Omega) = \text{constant} \quad (5.3)$$

After reorganizing, the objective function is chosen as:

$$F = -\max_{\Omega} \Phi \quad (5.4)$$

and is subject to two constraints:

1. Volume of the domain, $\Omega = \text{constant}$.
2. The Poisson equation with the specified boundary condition. (The PDE constraint)

5.2 Solution Technique

In order to solve this problem, the following items are needed:

- Representation of the geometry

The desired geometry consists of a base geometry plus a parameterized geometry. Functional cubic B-spline surfaces with nominal uniform knot set are used to parameterize the geometry change. Therefore the control points of the functional B-spline surface become the design variables in this optimization scheme.

- Mesh generator

A mesh generator is needed to produce mesh information for any suitable geometry. TETGEN [53] is chosen as the mesh generator. This

mesh generator cannot give a very refined mesh, therefore some special techniques are required to decrease the size of the mesh element.

- Poisson solver

A PDE solver is necessary to solve the Poisson equation with any acceptable meshes. The used solver is PUPS-1.0 [8]. It is a parallel Fortran code and has good performance.

- Optimization scheme

The optimization scheme is the core software in this problem. Although there are many optimization schemes suitable for this problem, such as the constrained and unconstrained optimization schemes, Lagrange-based methods, penalty based methods etc., an optimization scheme has been developed for this problem.

5.3 Theoretical results in shape representation

In this 3-dimensional optimization case, the geometry consists of the base geometry plus a geometry change. i.e.

$$z(\Gamma^U) = Y(\Gamma_{ini}) + \sum_{i,j=1}^n d_{i,j}^U N_{i,j}(u, v) \quad (5.5)$$

$$z(\Gamma^L) = Y(\Gamma_{ini}) + \sum_{i,j=1}^m d_{i,j}^L N_{i,j}(u, v) \quad (5.6)$$

where $d_{i,j}^U, d_{i,j}^L$ are the control points of the upper and lower B-spline surface respectively, $\Gamma_{ini}, \Gamma^U, \Gamma^L$ refers to the initial geometry, the upper geometry, the lower geometry respectively, and $N_{i,j}(u, v)$ are the base functions of the

B-spline. In this special case where the functional B-spline surfaces are in use, u, v can be regarded as the value of the x and y co-ordinates of points in the upper and lower geometry represented by B-spline surface respectively.

The functional B-spline surfaces are used to parameterize the geometry in order to decrease the possibility of getting a self-intersecting shape. If parametric B-spline surfaces are used as the methodology for shape representation, then the control points of the B-spline surface can go anywhere if there is no constraint to prevent them. Furthermore, for the parameterized B-spline surface, each control point is a vector of dimension 3. This will certainly increase the number of design variables and increase the possibility of getting a self-intersecting shape.

The expression for each of the two B-spline surfaces is given by the equation below:

$$z(u, v) = \sum_{i=1}^n \sum_{j=1}^m d_{i,j} N_{i,k}(u) N_{j,l}(v) \quad 0 \leq u \leq 1, \quad 0 \leq v \leq 1 \quad (5.7)$$

where the B-spline basis function is defined as equation 2.53:

The knot sets are $u_1, u_2, \dots, u_{n+k}, v_1, v_2, \dots, v_{m+l}$ and the order k, l are not specified yet.

Suppose a knot set is chosen as below:

$$(\underbrace{0, 0, \dots, 0}_k, \underbrace{\frac{1}{n-k+1}, \frac{2}{n-k+1}, \frac{3}{n-k+1}, \dots, \frac{n-k}{n-k+1}}_{n-k}, \underbrace{1, 1, \dots, 1}_k)$$

In this case, the volume is given by the equation below:

$$\begin{aligned} V(\Gamma) &= V(\Gamma^U) - V(\Gamma^L) \\ V(\Gamma^U) &= \int_S \Gamma_{ini}^U dS + \int_0^1 \int_0^1 (z^U(u, v)) du dv \\ V(\Gamma^L) &= \int_S \Gamma_{ini}^L dS + \int_0^1 \int_0^1 (z^L(u, v)) du dv \end{aligned}$$

where V represents the volume, $V(\Gamma), V(\Gamma_{ini}), V(\Gamma^U), V(\Gamma^L)$ represents the volume of the whole domain, the volume of the initial geometry, the volume of the upper part of the domain, the volume of the lower part of the domain respectively, provided that the shape consisting of the upper geometry Γ^U and the lower geometry Γ^L are not crossed. In the above equation array, z^U, z^L denote the upper and lower boundaries respectively.

Based on this equation array, the derivative of the volume with respect to the control points in the upper surface is given by the equation below:

$$\frac{\partial V}{\partial d_{i,j}^U} = \int_0^1 N_{i,k}(u) du \int_0^1 N_{j,l}(v) dv \quad (5.8)$$

where

$$\int_0^1 N_{i,k}(u) du = \frac{u_{i+k} - u_i}{k} \quad (5.9)$$

$$\int_0^1 N_{j,l}(v)dv = \frac{v_{j+l} - v_j}{l} \quad (5.10)$$

Hence, if a cubic B-spline curve with nominal uniform knot set is used, the gradient of the area with respect to the control points in the upper B-spline curve is given by

$$\frac{\partial V}{\partial d_{i,j}^U} = \left(\frac{u_{i+4} - u_i}{4} \right) \left(\frac{v_{j+4} - v_j}{4} \right) \quad (5.11)$$

For the control points in the lower B-spline surface, the gradient is given by:

$$\frac{\partial V}{\partial d_{i,j}^L} = - \left(\frac{u_{i+4} - u_i}{4} \right) \left(\frac{v_{j+4} - v_j}{4} \right) \quad (5.12)$$

and the volume is given by

$$V = \int_S \Gamma_{ini}^U dS - \int_S \Gamma_{ini}^L dS + \sum_{i=1}^n \sum_{j=1}^m (d_{i,j}^U \frac{u_{i+4} - u_i}{4} \frac{v_{j+4} - v_j}{4}) - \sum_{i=1}^n \sum_{j=1}^m (d_{i,j}^L \frac{u_{i+4} - u_i}{4} \frac{v_{j+4} - v_j}{4}) \quad (5.13)$$

where n, m are the number of control points in two directions.

By this geometry representation technique, the geometry constraint has been successfully solved, which then becomes a linear constraint.

5.4 Mesh considerations

The mesh generator used in the 3-dimensional case is TETGEN which generates tetrahedral meshes. The algorithms used in TETGEN are of the De-

launay type. For a 3-dimensional domain, defined by its boundary (such as a surface mesh), TETGEN generates the boundary constrained (Delaunay) tetrahedralization, conforming Delaunay tetrahedralization, and quality Delaunay mesh. Because of its limitation on the mesh size, the mesh directly generated by TETGEN is not refined enough. Therefore the scaling technique is used in the mesh generation.

For example, if a unit cube is inputted as the initial geometry, the first thing needs to be done is to scale it, say 100 times. Then the scaled geometry is used to generate the mesh elements. After the generation is finished, all of the coordinates in the mesh nodes need to be scaled back. However, the mesh connect information, boundary information, facets information can remain unchanged.

Second, for a very refined geometry, TETGEN has the problem of missing boundaries. More precisely TETGEN can produce a coarse mesh with no problems. However for a refined geometry, the boundary information produced by TETGEN are actually crossed. Therefore, the boundary information in the original code which is used to tell TETGEN the original boundary information must be preserved after the mesh generation.

In this geometric representation the desired geometry is a base shape plus functional B-spline surfaces. The methodology is needed in order to keep the connective information of the boundary.

For the functional B-spline surfaces, the coordinates at the Cartesian space are $u, v, z(u, v)$, i.e. u, v is the parameter value, represents the x, y coordinates of the functional B-spline surface. However for the arbitrary initial geometry, to build the connection between the point of B-spline surfaces and the point of the initial geometry in Cartesian space is difficult. Therefore the following methodology is used:

- points on functional B-spline surfaces $(u, v, z(u, v))$
- points on the initial geometry, if a sphere is used as an initial geometry, (r, θ, h) where h represents the height, θ is the angle, r is the radius.

By doing this, the x, y coordinates of the initial geometry is given by $r \cos \theta, r \sin \theta$ which determines the value of the parameters u, v of the functional B-spline surface. The coordinates of the designed geometry are then given by $(u, v, z(u, v) + h)$.

5.5 Other methods of solution

In this subsection, other possible methods for this 3-dimensional constrained optimization problem are discussed.

5.5.1 Penalty coefficient

The simplest method to deal with the constraints is to add penalty coefficients. However this requires the user to seek a balance between minimising the value of the objective function and conserving the constraints. If the

penalty coefficient is too big, then the value of the objective function cannot easily be decreased. On the other hand, if the penalty coefficient is too small, the constraints cannot be conserved. An example penalty function can be constructed as following:

$$M = \frac{F}{F^0} + 0.1 \times \frac{|Vol - Vol^0|}{Vol^0}$$

where $F = -\max_{\Omega} \Phi$, F^0 refers to negative maximum value for the initial geometry, and Vol , Vol^0 refers to the volume, the initial volume respectively. Now M becomes the new objective function. Instead of minimising $\frac{F}{F^0}$ keeping the volume constant, the function M is exactly the right function to be minimised and the constraint disappears.

5.5.2 Lagrange-based method

The Lagrangian (L) for this particular problem is:

$$L = F - \lambda_1 \int_{\Omega} (\nabla^2 \Phi - m) dV - \lambda_2 (V - m)$$

where λ_1, λ_2 are Lagrange multipliers, V is the volume of the domain Ω , and m is a constant, the value of the area of the initial shape is $\partial\Omega$ (or domain Ω). Since the main solution technique in PDE-constrained optimization is to solve the PDE in each optimization iteration the PDE constraints will be satisfied all the time. Then L changes to be:

$$L = F - \lambda_2 (V - m) \tag{5.14}$$

The basic idea of the Lagrange-based method is that a set consisting of $n+1$ real numbers:

$$(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{n-1}, \alpha_n, \lambda_2)$$

will finally be found (where $(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{n-1}, \alpha_n)$ are variables to describe the shape), such that

$$\left(\frac{\partial L}{\partial \alpha_1}, \frac{\partial L}{\partial \alpha_2}, \frac{\partial L}{\partial \alpha_3}, \dots, \frac{\partial L}{\partial \alpha_{n-1}}, \frac{\partial L}{\partial \alpha_n}, \frac{\partial L}{\partial \lambda_2} \right)^T = \mathbf{0} \quad (5.15)$$

It is straightforward that, $\frac{\partial L}{\partial \lambda_2} = 0$, will guarantee that the geometry constraint is satisfied. Furthermore, $\left(\frac{\partial L}{\partial \alpha_1}, \frac{\partial L}{\partial \alpha_2}, \frac{\partial L}{\partial \alpha_3}, \dots, \frac{\partial L}{\partial \alpha_n} \right)^T = \mathbf{0}$ will guarantee that the final shape is an optimum.

By using a Lagrange-based method, a constrained optimization problem is converted into an unconstrained optimization problem. The gradient of L with respect to $\alpha_i, i = 1, 2, \dots, n$ used in this method can be computed by the finite difference method, and the gradient of L with respect to λ_2 is given by the area change.

For practical use, it may be necessary to use other Lagrangians, such as the augmented Lagrangian given by the equation below:

$$L_A = F(\alpha) - \lambda_2 (V(\alpha) - k) + \frac{\rho}{2} (V(\alpha) - k)(A(\alpha) - k) \quad (5.16)$$

where $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)^T$.

Now we have the objective function and all of the design variables. The equation to compute the gradient of the volume with respect to the design variables (Equation 5.13) can be found in the previous section. The gradient of the objective function F with respect to the design variables can be computed by finite difference method. Therefore it is easy to start the optimization loop.

5.6 The developed method used in a 3-dimensional case

Since the objective function F is a function of state variable Φ and the shape $\partial\Omega$ ($F(\Phi, \partial\Omega)$), and the relationship between Φ and $\partial\Omega$ is determined by the Poisson equation with specified boundary condition ($\Phi(\partial\Omega)$), the state equation (the Poisson equation) can be first solved in each optimization iteration.

$$F(\Phi, \partial\Omega) \longrightarrow F(\Phi(\partial\Omega), \partial\Omega)$$

Second, because $\partial\Omega$ is a function of design variables, i.e., control points of the B-spline surfaces, ($\partial\Omega(\alpha_1, \alpha_2, \dots, \alpha_n)$, $\alpha_1, \alpha_2, \dots, \alpha_n$ are the control points of the B-spline surfaces), and the volume of the domain need to be fixed, it is a good choice to eliminate one of control points in order to keep the area constant. In this case, the constraint equation $V = m$ must be solved as well.

Now the problem is redefined as:

$$F = F(\alpha_1, \alpha_2, \dots, \alpha_{n-1}) \tag{5.17}$$

with α_n eliminated in each optimization iteration.

The justification for this method is given in the Appendix B. Based on this approach, it is clear that if α_n can be uniquely defined by $(\alpha_1, \alpha_2, \dots, \alpha_{n-1})$, then the solution is correct.

If the finite difference method is used to calculate the gradient, it is given by the equation below

$$\frac{\partial F}{\partial \alpha_i} \approx \frac{F(\alpha_1, \alpha_2, \dots, \alpha_{i-1}, \alpha_i + \epsilon, \alpha_{i+1}, \dots, \alpha_n + \delta) - F(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n)}{\epsilon} \quad (5.18)$$

where $i = 1, 2, \dots, n - 1$ and δ is given by solving the constraints:

$$\text{Area of } \Omega (\alpha_1, \alpha_2, \dots, \alpha_{i-1}, \alpha_i + \epsilon, \alpha_{i+1}, \dots, \alpha_n + \delta) = \text{constant}$$

To obtain the value of δ readers are refer to Section 5.3 for more details. It can be by using equation 5.13

5.6.1 CPU time

In this subsection, the analysis on how the CPU time distributes in the code is given. Since the developed code is integrated with two other software packages: the mesh generator TETGEN and the PDE solver PUPS-1.0, the CPU time consumed on these two must be taken into consideration as well. For the following table NMF and MF refers to no mesh refinement and with

mesh refinement respectively.

	Mesh time	Solve time	Time for one iteration
NMF	17.060	1.42	441.25
MF	159.996	11.395	11980.25

Table 17: CPU time distribution in 3-dimensional case

CPU time for File I/O

File description	File name	File size	CPU time
input geometry file	shape3d12.poly	337,703B	0.0452
output mesh files	3 files (.ele, .node, .face)	4,835,873B	3.349
input mesh file for PDE	sphere.grd	2,283,525B	2.157
output file from PDE solver	record3d1.out	53,910B	0.0377

Table 18: File I/O time in 3-dimensional case

- Note: 1. Second is chosen as the time unit, and byte is chosen as the size unit.
2. The mesh time and solve time are computed by the software; these values are obtained from their display message. When computing the CPU time for one iteration, the display message has been removed from the code.
3. The mesh time also includes the CPU time for File I/O.

5.7 Error estimation

Notation

- F the objective function
- $z_{i,j}$ the i, j th control point in the B-spline curve
- ϵ_{Δ} the triangulation error caused by linear approximation of a curve
- ϵ_r the residual error which is the norm of $\|AX - B\|$ in solving PDE

In order to analyse the error, it is necessary to define the error. The following term is defined, *root mean square error*.

$$J = \sqrt{\sum_{i=1}^n \sum_{j=1}^m \frac{(F_{i,j} - F_{i,j}^{exact})^2}{n * m}} \quad (5.19)$$

$(F_{i,j} - F_{i,j}^{exact})$ is regarded as the difference between the exact solution at the node i, j and the approximate one at the same node.

It is clear that the difference between the approximate value (value of objective function produced by code) and the exact one is influenced by the items below:

- the triangulation error because the elements are discretized
- the residual error of the PDE solver

5.7.1 Analysis of the influence of the residual error

The influence of residual error on the PDE solution is normally irrelevant when compared with the influence of the triangulation error, or the discretization error. To decrease the residual error, one usually sets a small value of the tolerance on the l_2 norm of the solution vector, say $1e - 10$, which would be possible for most of the current computers. In the 3-dimensional case, this argument is not true as indicated by the following content.

For a selected Poisson’s equation,

$$\frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\partial^2\Phi}{\partial z^2} = -1.0 \quad in \quad \Omega \tag{5.20}$$

with the boundary condition

$$\Phi = 0 \quad on \quad \partial\Omega$$

where Ω is a unit sphere, the centre is (1.0, 0.0, 0.0). We have the analytical solution:

$$\Phi = \frac{1.0}{6.0} - \frac{(x - 1.0)^2 + y^2 + z^2}{6}$$

Therefore it is convenient to compute the error now.

Choice of the tolerance	Iterations	Error in solution	Sum
1e-3	19	0.00120975	0.10669
1e-4	26	0.00120957	0.106657
1e-5	31	0.00121013	0.106756
1e-6	37	0.00121011	0.106753
1e-7	41	0.00121011	0.106752
1e-8	48	0.00121011	0.106752
1e-9	53	0.00121011	0.106752
1e-10	58	0.00121011	0.106752

Table 19: Residual error investigation in 3-dimensional case

As indicated by the above table, a tolerance of $\tau = 1e - 7$ is a sufficiently good choice because there is no improvement on the error or sum of the error when decrease the value of the tolerance. Hence there is no need to put too small a value on τ .

5.7.2 Analysis of the influence of triangulation error

In order to analyse the influence of triangulation error, the shape cannot be described by B-spline surfaces, otherwise the influence of least square error will automatically be included.

Now the *sphere* is the input shape (not the shape approximated by B-spline surfaces), the number of boundary points (all of the boundary points are evenly distributed) will be doubled to see how the solution changes.

The following table reflects the influence of triangulation error on the PDE solution:

Number of boundary points	Number of nodes	Error in solution	Rate
40	1658	0.01631362	8.02
80	10539	0.00204123	
40	814	0.00216005	8.06
80	6700	0.000285412	

Table 20: Triangulation error investigation in 3-dimensional case

From this table, it is obvious that when mesh size decreases by about 1/2, the error in the solution will decrease by 1/8 as well.

The error analysis here is based on the real sphere not on the approximated sphere consisting of B-spline surfaces since the influence of the least square error is already excluded.

The conclusion for triangulation error is: if the numbers of boundary points

is doubled, and at the same time increasing the total number of nodes, (to make the mesh size decrease a half) then the triangulation error will decrease cubically.

Even if the sphere points have been mapped into the functional B-spline surface points, the rule still exists.

Number of elements	Number of nodes	Error in solution	Rate
5680	1674	0.0162715	
37658	10314	0.0019668	8.02

Table 21: Triangulation error investigation in 3-dimensional case (functional B-spline surface case)

5.7.3 Conclusion on error analysis

From the above, the main error in the developed code is caused by triangulation error. It has the influence of residual error, but it can be decreased and in fact this influence is quite small. For the triangulation error, it can be decreased by increasing the number of boundary points and distributing these points evenly.

5.8 Results

For the different initial shapes, the new method starts from 16 free control points, and the initial mesh has about 40,000 tetrahedra. When the knot insertion algorithm is required, an additional knot is uniformly inserted into the existing one, and the number of control points becomes 81. When the mesh refinement starts, the fine mesh is produced which has about 200,000

tetrahedra.

The middle point in the lower B-spline surface is chosen as the special point to keep the constraint, i.e. keeping the area constant. When the knot insertion algorithm starts, in order to be consistent, the special point should be selected to be the middle point in the new lower B-spline surface. If the number of control points for the new lower B-spline surface is a even number, then it will not damage the optimization no matter which middle point is selected as the special point. This has been tested.

The idea of the method is presented earlier in this thesis and there are different ways to fulfil the idea. For example, one can locally insert knot to achieve the requirements on the geometry instead of inserting knots uniformly.

The criteria to start the knot insertion algorithm is set as $|F_{k+1} - F_k| < \eta$. The initial shape is given by the following two pictures

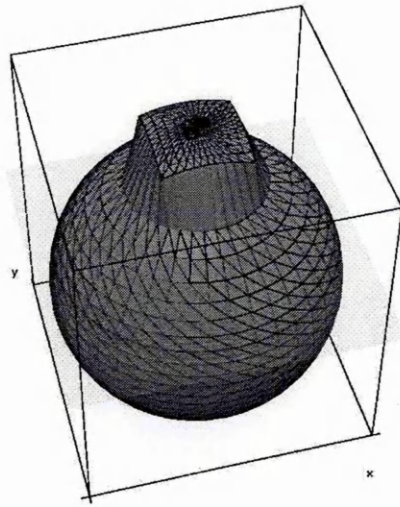


Figure 33: Initial shapes (top view)

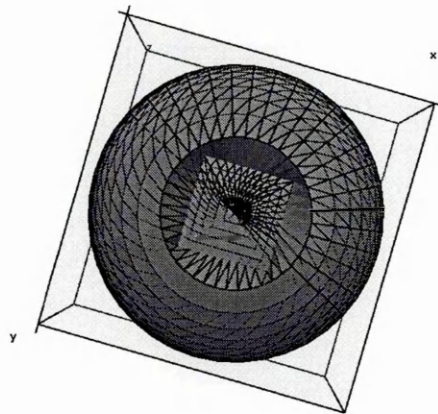


Figure 34: Initial shapes (bottom view)

These two pictures are from the same geometry but viewed from different angles. One is the top view, the other is the bottom view. The optimal shape is also given in Fig. 35, Fig. 36 , with the bottom view and the top view.

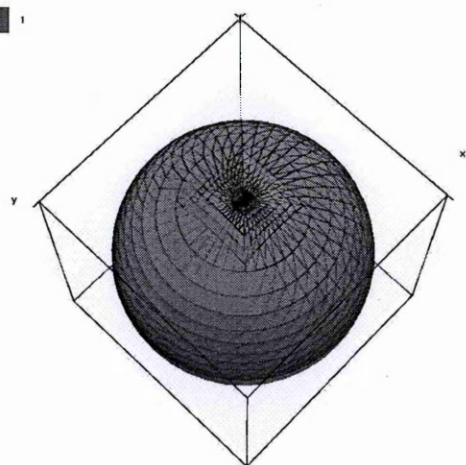


Figure 35: Optimal shapes (top view)

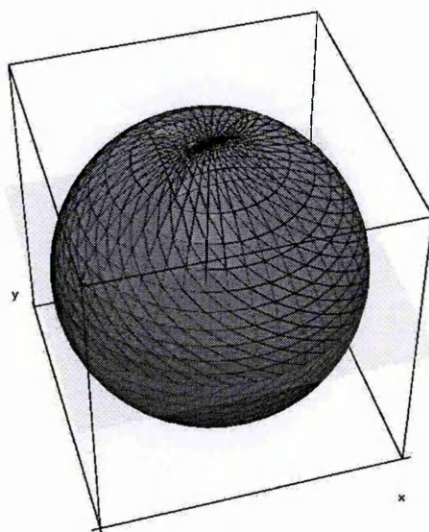


Figure 36: Optimal shapes (bottom view)

The other two optimal shapes are given in Fig. 37, Fig. 38. The optimal geometry below is achieved with a weak end condition.

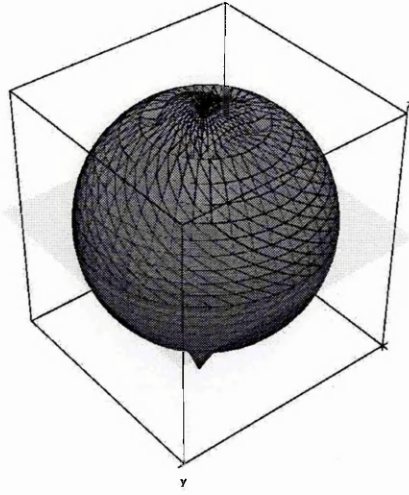


Figure 37: Optimal shapes (top view)

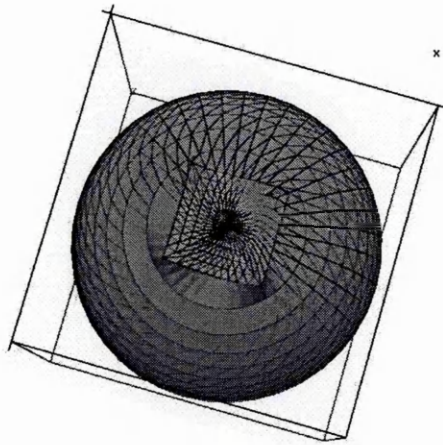


Figure 38: Optimal shapes (bottom view)

The optimal geometry given in Fig. 39, Fig. 40, is achieved by changing the value of η .

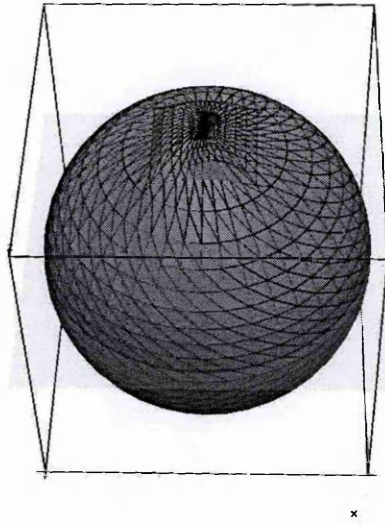


Figure 39: Optimal shapes (top view)

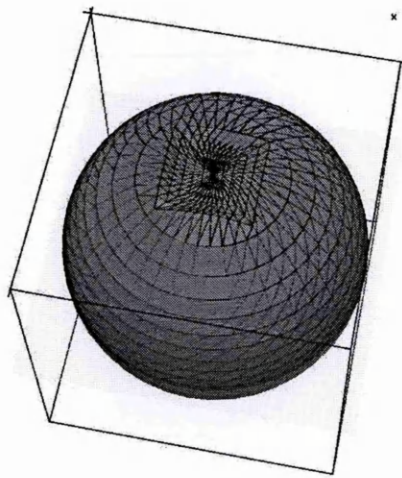


Figure 40: Optimal shapes (bottom view)

5.9 Summary

The developed Super-Reduced has been tested in the 3-dimensional case. It has been shown that this method can converge to an optimal solution.

The geometry representation and the mesh issue are discussed detail. The designed geometry used in the tested cases consists of the base geometry plus a geometry change. The geometry change is parameterized by functional B-spline surfaces, therefore potentially decrease the possibility of getting self-intersecting shapes. Furthermore, it will decrease the number of the control points, i.e, the number of the design variables.

TETGEN is selected as the mesh generator, which has a problem of missing boundary for refined mesh. Therefore the boundary information must be preserved after the mesh generator. However, this will inevitably increase the memory requirements.

The final results show the convergence of the method. Because the intensively computational cost, (for a mesh of around 0.1 million elements, it takes about 3.5 hours for one optimization iteration!) the investigation of 3-dimensional case is not so extensive.

6 Conclusion and Future work

In this thesis, the mathematical background, the details of the new optimization method using a knot insertion algorithm and the B-spline shape functions have been described.

Current research on geometry optimization focuses on the methods of computing the gradient and the sensitivities. Geometric representation has been somewhat neglected. The traditional shape function remains a useful method to describe the geometry. However, this is an algebraic way of describing geometry. Although the Bézier curves and B-splines have been employed to parameterize the geometry in some cases, it is still not mainstream in this field.

The present research combines the B-spline as shape functions and knot insertion. The purpose of doing so is to decrease the number of optimization iterations and produce an improved estimated optimal value and associated geometry. The developed method is BFGS-based, because of its quadratic convergence when the point is sufficiently close to the optimal point. Also, the limitations of the BFGS method are partly solved by the new method. The BFGS's weakness lies in its matrix operation because it requires the matrix update according to the BFGS formula in each optimization iteration. Therefore if the design space is large then the computational cost will be high. The new method allows the initial number of design variables to be kept to a minimum. Hence, although the matrix operation is still required,

it is limited to a small dimension for much of the calculation.

The present work on the geometry optimization method is believed to be original. The following features have been implemented:

- The knot insertion algorithm has been firstly incorporated into the geometry optimization method.

This is the first time knot insertion algorithm has been integrated with the geometry optimization method, see Chapter 3.

- The super-reduced idea

Although the super-reduced idea has been implemented in others' work [62], this is an independent research on this idea in different applications. Furthermore, this report describes a sufficient condition of using the super-reduced idea, see Appendix B.

PDE-constrained optimization problems with an equality geometry constraint are described. A robust novel method with B-splines as shape functions, the knot insertion algorithm, super-reduced idea, and mesh refinement have been developed to solve this class of problem. The basic idea of the developed new method is to first convert the constrained optimization problem into an unconstrained one and when the shape is close to the optimum one, to use mesh refinement to improve the accuracy of the gradient vector. Introducing a way of dealing with self-intersecting shapes is also investigated. More details of the method can be found in Chapter 3. The main feature and results of the developed algorithm on a number of test cases have been presented.

- Efficiency.

Tests have shown that the newly developed method is better than the BFGS method in geometry optimization and the EXTREM method, see Sections 4.3, 4.7, 4.8. The new method can find the optimum shape within less optimization iterations, and less CPU time.

- Preservation of constraints.

In the developed method, the constraint is maintained all the time so that all changes are feasible. This is a consequence of the super-reduced idea.

- Dealing with self-intersecting shapes.

The self-intersecting shape is not an issue in practical problems because in that case, the shape seldom changes very much and hence it is unlikely that a self-intersecting shape will occur. However, in test cases, the shape can be self-intersecting because the control points are totally free. An algorithm to prevent and deal with self-intersecting shapes is therefore necessary and has been developed, see Section 3.6.

- Accessing the analytical value.

Some experimental investigation of error has been done. Based on the conclusion of the error analysis, it is possible to approach the analytical value as long as the computational cost is affordable, see Section 4.4.

The new method can decrease the computational cost. It gives better results when B-splines are used to parameterize the geometry.

In summary, the main contribution of the current work is: a new method

for geometry optimization with B-splines as shape functions, the knot insertion algorithm, super-reduced idea, and mesh refinement has been developed. The method has been tested in a number of cases and shown to be more efficient than the BFGS method and the EXTREM method in these cases in terms of the CPU time, number of function calls, number of the optimization iterations.

In addition to this, the use of high order B-splines is investigated, but not in great detail. This could be very useful for the future research. We have indicated that how to use the high order B-spline to improve the efficiency (saving CPU time). Beside this a multi-objective optimization method is also developed for geometry optimization problems.

Future possible enhancement for the new method is listed below:

- Developing a local knot insertion algorithm based on the geometry properties.

A local knot insertion algorithm would be necessary if there were some special requirements in the geometric properties. In this case, the global knot insertion algorithm would not be as efficient as a local knot insertion algorithm which can insert a knot where it is necessary.

- Integrating some other gradient evaluation methods.

The current method to evaluate the super-reduced gradient is a finite difference type. This type is not the best choice for the situation where there is a large design space. Both the adjoint method and the reverse

mode of the automatic differentiation are particularly efficient on the problem where there are a lot of design variables.

The automatic differentiation depends on the code type. However, there are current codes for geometry optimization of different types. The PDE solver is a mixture of FORTRAN 90 and C code, mainly written in FORTRAN; the optimization scheme is pure C++, the mesh generator is written in C language. Therefore, it is almost impossible to use automatic differentiation on the current optimization software. If automatic differentiation is used, it is desirable that all software involved is the same type. Therefore, some of the software will be replaced and obviously this requires the updates of the interfaces.

The adjoint method is particularly good for problems which have many design variables but fewer constraints. If the adjoint method is used, then the gradient can be obtained by solving one adjoint equation and one PDE in one optimization iteration. However, this will bring the complexity to the code. Furthermore, if the objective function changes, then a new adjoint equation must be developed.

- Integration some of other constrained optimization methods.

At this stage, the new method is developed for dealing with the constrained optimization problems with equality constraints. In multi-objective optimization cases, a SQP-based method is also developed. However, as no methods are good to deal with general non-linear con-

strained optimization problems, some other optimization methods could be included; also an idea can be further developed to decide in which cases to start which methods.

- Parallelize the optimization methods.

In the newly developed optimization methods, the PDE solver is a parallelized routine, in order to decrease the CPU time, although the optimization are not applicable to be parallelized in general. The methods to compute the gradient vector can be parallelized if the finite difference method is used.

- Extend the application to other governing equations.

The current used governing equations are Poisson's equation and the full potential equation with boundary analysis, although the developed optimization method is independent of governing equations. Obviously for the non-linear governing equations, they will be computationally expensive to solve. Future applications may include the RANS (Reynolds-Averaged Navier-Stokes) equation or even LES (Large Eddy Simulation).

- Extend the application to some special geometry.

In industrial cases, smooth geometry is normally the primary choice. However, in some special cases, it may be necessary to use continuous but non-smooth geometry, or there may exist some special requirements on the geometry, say the curvature. This should be under consideration for future applications as well.

- Using p-refinement or h-p refinement to refine the mesh.

The currently used mesh refinement is a local h-mesh refinement. This means only the mesh size has been decreased. However, for smooth problem p-mesh refinement seems to be a good choice because this technique does not change the elements, rather it introduces more node points to each system of equation. Furthermore, it is also a good choice to use h-p mesh refinement because this can decrease the computational cost.

References

- [1] Anderson, J.D., *Computational fluid dynamics: the basics with applications*, New York: McGraw-Hill, 1995.
- [2] Anderson, W.K., Venkatakrishnan, V., *Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation*, AIAA report 97-0643, 1997.
- [3] Anderson, W.K., Nielsen, E.J., Sensitivity analysis for Navier-Stokes equations on unstructured meshes using complex variables, *AIAA journal* Vol. 39, No.1, pp 56-63, 2001.
- [4] Arvo J.,(editor), *Graphics gems II*, Academic press, ISBN: 0120644819. 1991.
Also available online: <http://www.acm.org/pubs/tog/GraphicsGems/>
lastest access time 20/05/2005.
- [5] Baysal, O., and Ghayour, K., *Continuous adjoint sensitivities for general cost functionals on unstructured meshes in aerodynamic shape optimization*. AIAA-1998-4904, 1998.
- [6] Beale, E.M.L., *Introduction to optimization*, ISBN 0-471-91761-3, Wiley, 1988

- [7] Becker, D., Thompson, C.P., *A novel, parallel PDE solver for unstructured grids*, In 5th International Conference on Large-scale Scientific Computation, 2005.
- [8] Becker, D., *Parallel unstructured solver for linear partial differential equations*, Ph.D. thesis of Cranfield University, 2006.
- [9] Bischof, C., Carle, A., Corliss, G., Griewank, A. and Hovland, P., AD-IFOR: Generating derivative codes from fortran programs, *Scientific Programming*, Vol 1, No.1, pp 11-29, 1992.
- [10] Borggaard, J., Burns, J., *A sensitivity equation approach to shape optimization in fluid flows*, ICASE report NO.94-8, 1994
- [11] Burger, M., A framework for the construction of level set methods for shape optimization and reconstruction, *Interface and free boundaries*, Vol 5, pp 301-329, 2003.
- [12] Burkardt, J., Gunzburger, M. Sensitivity discrepancy for geometric parameters, *ASME CFD for design and optimization*, Vol. 232, pp 9-15, 1995.
- [13] Choi, T.J., *CFD shape optimization based on an adjoint variable formulation of the compressible Navier-Stoke equations*, 1999 graduate student conference, Carnegie Mellon university, 1999.
- [14] Coleman, T.F., Moré, J., Software for estimating sparse Jacobian matrices *ACM transaction on mathematical software*, Vol 10, No. 3, pp 329-345 September 1984.

- [15] Curtis, A., Powell, M., and Reid, J., On the Estimation of Sparse Jacobian Matrices, *J. Inst. Math. Appl.* Vol. 13, pp 117-119, 1974.
- [16] Desideri, J.A., and Janka, A., *Multilevel shape parameterization for aerodynamic optimization - application to drag and noise reduction of transonic/supersonic business jet* European congress on computational methods in applied sciences and engineering, ECCOMAS 2004, Jyväskylä, 24-28 July 2004.
- [17] Doorly, D.J., Peri6, J., *Supervised parallel genetic algorithms in aerodynamic optimization*, AIAA paper 97-1853, 1997.
- [18] Farin, G.E., *Curves and surfaces for CAGD: a practical guide 5th*, ISBN 1558607374, San Diego: Academic press, 2002.
- [19] Fletcher, C.A.J., *Computational techniques for fluid dynamics Vol. 1*, Berlin : Springer Verlag, 1988.
- [20] Fletcher, R., *Practical methods of optimization second edition*, ISBN 0-471-91547-5, Wiley, 1987
- [21] Griewank, A., Juedes, D., and Utke, J., ADOL-C: A package for the automatic differentiation of algorithms written in C/C++, *ACM*, Vol 22, No.2, pp 131-167, 1996.
- [22] Giles, M.B., Pierce, N.A., An introduction to the adjoint approach to design, *Flow, turbulence and combustion*, Vol. 65(3-4):pp 393-415, 2000.

- [23] Giles, M.B., Duta, M.C., Muller, J.D., and Pierce, N.A., Algorithm developments for discrete adjoint methods, *AIAA Journal*, Vol. 41, No. 2, 2003.
- [24] Hicks, R. M., Henne, P.A., *Wing design by numerical optimization*, AIAA 77-1247, AIAA Aircraft System and Technology Conference, Seattle, Wash., Aug. 22-24 1977.
- [25] Hicks, R. M., Vanderplaats, G.N., *Application of numerical optimization to the design of supercritical airfoils without drag -creep*, SAE paper 770440, 1977.
- [26] Hicks, R.M., Vanderplaats, G.N., Murman, E.M., King, R.R., Airfoil section drag reduction at transonic speeds by Numerical optimization, *SAE paper 760477*, 1976.
- [27] Holst, T.L., Pulliam, H., *Aerodynamic shape optimization using a real-number-encoded genetic algorithm*, AIAA 2001-2473, 2001
- [28] INRIA, BAMG v0.68, <http://www-rocq1.inria.fr/gamma/cdrom/www/bamg/eng.htm> latest access time 20/05/2005.
- [29] Iollo, A., Kuruvila, G., Ta'asan, S., *Pseudo-time method for optimal shape design using the Euler equations*, ICASE report no. 93-78, 1993
- [30] Iollo, A., Salas, M.D., *Optimum Transonic airfoils based on the Euler Equations*, ICASE Rep. No. 96-76, 1996.
- [31] Jameson, A., Aerodynamic design via control theory, *Journal of scientific computing*, Vol 3, No.3, 1988.

- [32] Jameson, A., *Optimum aerodynamic design Using CFD and control theory*, AIAA paper 95-1729, June 1995.
- [33] Jacob, H.G., *Rechnergestutzte optimierung statischer and dynamischer system*, Springer, Verlag, 1982.
- [34] Kim, I.Y. and Weck, O.L. *Adaptive weighted-sum method for bi-objective optimization: Pareto front generation*, Struct Multidisc Optim, Vol. 29, 2005
- [35] Kuruwila, G., Ta'asan, S., Salas, M.D., *Airfoil design and optimization by the one-shot method*, AIAA 95-0478, 1995
- [36] Lee, K.D., Eyi, S., *Transonic airfoil design by constrained optimization*, AIAA 91-3287, AIAA 9th applied aerodynamics conference, Baltimore, MD, 1991
- [37] K.D.Lee, S. Eyi, *Transonic airfoil design by constrained optimization*, Journal of aircraft, Vol. 30, No.6, 1993.
- [38] Lohner, R., *Applied CFD techniques: an introduction based on finite element methods*, ISBN 0471498432, John Wiley, 2001.
- [39] Mohammadi, B., Pironneau, O., *Applied optimal shape design*, Scientific and Engineering computations for the 21st century. Proc 15th Toyota conference. North Holland, 2002
- [40] Mohammadi, B., Pironneau, O., *Applied shape optimization for fluids*, ISBN 0198507437, Oxford: Clarendon Press, 2001.

- [41] Muyl, F., Dumas, L., Herbert, V., Hybrid method for aerodynamic shape optimization in automotive industry, *An International Journal Computers and Fluids*, vol. 33, no. 5, June 2004
- [42] Osher, S.J., Santosa, F., Level set methods for optimization problems involving geometry and constraints I. Frequencies of a two-density inhomogeneous drum *Journal of Computational Physics*, Vol. 171 , Issue 1, pp 272 - 288, July 2001
- [43] Osher, S., and Sethian, J.A. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations *Journal of computational physics*, Vol 79, pp. 12-49, 1988.
- [44] Pironneau, O., On optimum profiles in Stokes flow, *Journal of fluid mechanics*, Vol.59, 1973
- [45] Pironneau, O., On optimum design in fluid mechanics, *Journal of fluid mechanics*, Vol.64, 1974
- [46] Polak, E., *Optimization: algorithms and consistent approximations*, Springer, New York, 1984
- [47] Preparata, F.P., Shamos, M.I., *Computational geometry an introduction*, ISBN 0-387-96131-3, Springer-Verlag New York inc., 1985
- [48] Qiao, Z., A PC computer system for the design and numerical computation of advanced airfoils, *A biography on national applied computer software*, P.R. China, Vol. 5, 1988

- [49] Rappoport, A., Sheffer, A., Bercovier, M., Volume-preserving free-form solids, *IEEE transactions on visualization and computer graphics*, Vol.2, No.1, 1996.
- [50] Rostaing, N., Dalmas, S. and Galligo, A., *Automatic differentiation in Odyssee*, Tellus, 45A, 1993.
- [51] Saad, Y., *Iterative methods for sparse linear systems.*, PWS Publishing company, Boston, 1995
- [52] Samareh, J.A., *A survey of shape parameterization techniques*, CEAS/AIAA/ICASE/NASA Langley international forum on aeroelasticity and structural dynamics 1999, Williamsburg, Virginia, June 22-25, 1999.
- [53] Si, H., *On Refinement of Constrained Delaunay Tetrahedralizations*, In proceedings of the 15th international meshing roundtable, 2006.
- [54] Sobieski, J.S., *The case for aerodynamic sensitivity analysis*, Presents at the NASA/VPI and SU Symposium on Sensitivity Analysis in Engineering, Sept. 25-26, 1986.
- [55] Soemarwoto, B.I., *The variational method for aerodynamic optimization using the Navier-Stokes equations*, ICASE report no. 95-32, 1995
- [56] Squire, W., Trapp, G., Using complex variables to estimate derivatives of real functions, *SIAM review* Vol. 10, No. 1, pp 110-112, 1998.
- [57] Ta'asan, S., *Introduction to shape design and control*, Lecture series, Carnegie Mellon University, 2002.

- [58] Ta'asan, S., *Pseudo-time methods for constrained optimization problems governed by PDE*, ICASE report No 95-32, 1995
- [59] Wah, B.W., Chen, Y., *Constrained genetic algorithms and their applications in non-linear constrained optimization*, 12th International conference on tools with artificial intelligence, November 2000.
- [60] Vetsa, V.N., Computation of sensitivity derivatives of Navier-Stoker equations using complex variables, *Advances in engineering software*, Vol 31, No 8, 2000
- [61] Winter, G., Périaux, J., Galán, M. and Cuesta, P. Genetic Algorithms in Engineering and Computer Science (EUROGEN95), Las Palmas (Spain), 1995.
- [62] Xie, L., *Gradient-based optimum aerodynamic design using adjoint methods*, PhD thesis of Virginia Polytechnic Institute and State University, 2002
- [63] Zhong, B., Qiao Z., *Multiobjective optimization design of transonic airfoils*, ICAS-94-2.1.1, 1994

A Publications during the Ph.D study

Sherar, P.A., Thompson, C.P., Xu, B., Zhong, B., *An Optimization Method Based On B-spline Shape Functions & the Knot Insertion Algorithm*, World Congress of Engineering, London, 2007.

Received "the second best paper award", recommended to journal and the chapter of the book

Xu, B., Zhong, B., Thompson, C.P., Sherar, P.A., *Multiobjective Optimization Design of Transonic Airfoils Using A New Method*, EUCASS, 2007.

Sherar, P.A., Thompson, C.P., Xu, B., Zhong, B., A Novel Shape Optimization Method Using Knot Insertion Algorithm in B-Spline and Its Application to Transonic Airfoil Design [submitted], *Computer Methods in Applied Mechanics and Engineering*, 2007

Xu, B., Zhong, B., Thompson, C.P., Sherar, P.A., Multiobjective Optimization Design for Transonic Airfoils Based on SQP and the Knot Insertion Algorithm of B-spline [to be submitted], *International Journal for Numerical Methods in Engineering*, 2007.

B Mathematical justification for super-reduced-method

Suppose we have a constrained optimization problem with one equality constraint. The objective function is $F(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n)$ where $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n$ are n design variables defined on R^n . One constraint is $c(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n) = 0$, which is not specified clearly yet. We want to prove a theorem below:

If there exists a unique solution in the constraint equation for one of design variables, say α_i , (that means $\alpha_i = g(\alpha_1, \alpha_2, \dots, \alpha_{i-1}, \dots, \alpha_{i+1}, \dots, \alpha_n)$), then the local minima for reduced system are also local minima for ordinary system

In the above theorem, the reduced system means we convert (let us choose $i = n$ which will not damage the generality but with short expression):

$$F(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n) \longrightarrow F(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{n-1}, g(\alpha_1, \alpha_2, \dots, \alpha_{n-1}))$$

The right term in the above expression is the reduced system, note it has $n - 1$ design variables only. So if we use this system to do an optimization, then we will get one local minimum:

$$(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{n-1})$$

and the α_n is obtained by solving the constraint equation $c(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n) = 0$.

The ordinary system means we construct a Lagrangian function for this problem.

$$L(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n, \lambda_1) = F(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n) - \lambda_1 c(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n) \quad (\text{B.1})$$

For this system, we want to find a set of

$$(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n, \lambda_1)$$

which will make:

$$\frac{\partial L}{\partial \alpha_1}, \frac{\partial L}{\partial \alpha_2}, \frac{\partial L}{\partial \alpha_3}, \dots, \frac{\partial L}{\partial \alpha_n}, \frac{\partial L}{\partial \lambda_1} = 0 \quad (\text{B.2})$$

We now need to prove that the local minima for the reduced system is also satisfied the above equation.

Proof

Let us simplify the ordinary system first. Because $c(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n) = \alpha_n - g(\alpha_1, \alpha_2, \dots, \alpha_{n-1})$ The first order necessary condition (equation B.2) becomes:

$$\begin{pmatrix} \frac{\partial F}{\partial \alpha_1} - \lambda_1 \frac{\partial c}{\partial \alpha_1} \\ \frac{\partial F}{\partial \alpha_2} - \lambda_1 \frac{\partial c}{\partial \alpha_2} \\ \vdots \\ \frac{\partial F}{\partial \alpha_{n-1}} - \lambda_1 \frac{\partial c}{\partial \alpha_{n-1}} \\ \frac{\partial F}{\partial \alpha_n} - \lambda_1 \\ c(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n) \end{pmatrix} = 0 \quad (\text{B.3})$$

In the above equation, the second last term is a bit different because $c(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n) = \alpha_n - g(\alpha_1, \alpha_2, \dots, \alpha_{n-1})$ (then $\frac{\partial c}{\partial \alpha_n} = 1$).

In the reduced system, the first order necessary condition is:

$$\begin{pmatrix} \frac{\partial F}{\partial \alpha_1} \\ \frac{\partial F}{\partial \alpha_2} \\ \vdots \\ \frac{\partial F}{\partial \alpha_{n-1}} \\ \alpha_n - g(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{n-1}) \end{pmatrix} = \mathbf{0} \quad (\text{B.4})$$

Now it is very clear if we choose $\lambda_1 = \frac{\partial F}{\partial \alpha_n}$ then all of equalities in the equation will automatically satisfy all of the equalities in the last equation.

C Solution to the Poisson equation

In order to get an idea of the accuracy and efficiency of the Poisson solver, we use two different Poissons equations with known analytical solution. The first is

$$\nabla^2 \Phi = -2\pi^2 \sin(\pi x) \sin(\pi y) \quad \text{in } \Omega \quad (\text{C.1})$$

with the boundary condition

$$\Phi = 0 \quad \text{on } \partial\Omega$$

and Ω is a rectangular from $(-1, -1)$ to $(1, 1)$ The exact solution for such a problem is

$$\Phi = \sin(\pi x) \sin(\pi y)$$

For a mesh with 125 nodes , the average error for each node is around $2e-2$ and the squared error is 0.299578

The second sample is our problem:

$$\nabla^2 \Phi = -1.0 \quad \text{in } \Omega \quad (\text{C.2})$$

with the boundary condition

$$\Phi = 0 \quad \text{on } \partial\Omega$$

where Ω is a circle. (centre: (0.5,0), radius: 0.5) The exact solution for this problem is

$$\Phi = \frac{0.250 - (x - 0.5)^2 - y^2}{4}$$

For a mesh with 1235 nodes, the squared error is 0.00724796. If we increase the number of nodes, the average error in each nodes decrease. We use a very fine mesh to get the gradient information.

D Data collection with mesh generator BAMG

Due to the fact that the mesh generator MESH2D has problems working with the parallel computer, we use another mesh generator to replace it, which is BAMG, from INRIA lab, France. By using this software, we includes one special point into our mesh file, the geometry centre, which is suspected to be the maximum for our special PDE-constrained optimization problem. An

algorithm to compute the centre can be found in [4]. Later, the data we get proves this when the shape is close to the approximated circle. We use 0.062286644842548415 on $(0.5, 1.33781e - 20)$ as the maxima for the shape before perturbing any control points. The step size is chosen as 0.001. The gradient information and other necessary information are listed in the table below.

No.	gradient	value (F^c)	No.	gradient	value
2	1.64e-3	0.06228828408431918	20	1.78e-3	0.06228842561814151
3	3.27e-3	0.06228991805343941	19	3.59e-3	0.062290230551914474
4	5.12e-3	0.062291770280052914	18	5.34e-3	0.06229198433953233
5	4.03e-3	0.062290672721254124	17	4.17e-3	0.062290812431539714
6	3.67e-3	0.062290313596206216	16	5.6e-3	0.062292252611184845
7	2.21e-3	0.06228885178840413	15	5.44e-3	0.06229209233419361
8	4.43e-3	0.0622910793508614	14	4.37e-3	0.06229101296318704
9	4.11e-3	0.062290758254178506	13	3.85e-3	0.06229049841678154
10	4.32e-3	0.06229096500108377	12	3.84e-3	0.06229047994720484
11	4.61e-3	0.06229125611725885			

E The second best paper award paper: An Optimization Method Based On B-spline Shape Functions & the Knot Insertion Algorithm

Abstract—A new method is presented to deal with shape optimization problems. In this method, the geometry is parameterized by B-spline shape functions with the control points of the B-spline curves becoming the design variables in the optimization scheme. The core idea of the

method presented is to introduce the knot insertion algorithm which can keep the geometry unchanged whilst increasing the number of control points. Besides this core idea, the super-reduced method and mesh refinement are also employed. The super-reduced method is used to get rid of the equality constraints. Hence a constrained optimization problem is converted into an unconstrained optimization problem with the constraints kept unchanged. We apply the method to two applications; first a problem involving Poisson's equation, and second, an application of the method to an airfoil design problem based on the airfoil NACA 0012. In both of these applications, the new method is shown to be efficient compared with 'standard' methods. In particular, in the airfoil problem we obtain a CPU saving time of about 42% compared with the EXTREM method. *Keywords: B-spline, knot insertion algorithm, mesh refinement, optimization*

Introduction

Shape optimization problems can be regarded as a problem of finding a shape which can achieve a given performance while satisfying some constraints. Two kinds of method are used in optimization; gradient-based methods and global methods. The gradient-based method is efficient in finding a local optimum while the global method has the advantage of finding a global optimum and does not require the objective function to be differentiable.

With the gradient-based method, the gradient information is necessary to

minimize the objective function. In shape design problems, to obtain the gradient of the objective function with respect to the design variables one usually needs the gradient of the state variables with respect to design variables, called the sensitivity. Some well-known methods to compute the sensitivity are the finite difference method [16,17], the complex variable method [2], automatic differentiation [4,5], sensitivity analysis [6,7], and the adjoint method [12,13,14].

In shape optimization geometry representation is an important issue. Hicks, Henne, Vanderplaats [10,11] use shape functions to parameterize shape changes. Another approach is to use the coordinates of every surface point as design variables. The approach removes the geometric model from the optimization loop, however, it may lead to discontinuities in the gradient which can be eliminated by a smoothing technique. An example of the use of B-spline control points to parameterize the geometry is given by Anderson and Venkatakrishnan [1].

In this paper, B-splines are chosen as shape functions to parameterize the changing geometry. The knot insertion algorithm has been incorporated into the shape optimization. Using this technique, the initial number of control points can be decreased. We then use the BFGS-based gradient method to find the optimum shape.

Method Description

In this method, the geometry is parameterized by B-spline shape functions with nominal uniform knot set. Therefore, the control points of the B-spline curves become the design variables in the optimization scheme. The knot insertion algorithm is introduced to keep the geometry unchanged whilst increasing the number of control points. The initial number of control points can be set to a small number, potentially decreasing the number of optimization iterations. To achieve a suitable accuracy the final number of control points can be very large for the value of the objective function. In the approach taken here the knot insertion algorithm kicks in when the solution is close to the optimum solution. After that, a large number of design variables will be produced as specified by the user. Since the knot insertion algorithm does not change the geometry, the optimization problem will be kept the same. Beside this core idea, the super-reduced method and mesh refinement are also incorporated. The super-reduced idea [19] is employed to convert the constrained problem into an unconstrained one (where the constraints are kept unchanged all the time). Using this method the gradient becomes a super-reduced gradient. Mesh refinement is extensively used in shape optimization. This approach is very useful when the exact gradient is difficult to obtain. In doing so, most of iterations will be done on the coarse grid and with relatively few on the refined grid, thus making the method more efficient.

In the new method, the selection of the search direction and the update of the approximate Hessian matrix follows the same rule as BFGS method

[9] does:

$$\begin{aligned} \mathbf{s}_k &= -H(\mathbf{x}_k)^{-1} \mathbf{g}(\mathbf{x}_k) \\ H_{k+1} &= H_k + \frac{\mathbf{q}_k \mathbf{q}_k^T}{\mathbf{q}_k^T \mathbf{p}_k} - \frac{H_k \mathbf{p}_k \mathbf{p}_k^T H_k}{\mathbf{p}_k^T H_k \mathbf{p}_k} \end{aligned}$$

where $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{q}_k = \mathbf{g}(\mathbf{x}_{k+1}) - \mathbf{g}(\mathbf{x}_k)$, and $\mathbf{x}_k, \mathbf{g}_k$ denotes the point and the gradient at the k -th iteration respectively.

Knot insertion algorithm

The knot insertion algorithm [8] allows one to insert a knot into a B-spline curve without changing its shape. After knot insertion, the B-spline evaluation ($\mathbf{p}(u)$) function changes from

$$\mathbf{p}(u) = \sum_{i=1}^n \mathbf{d}_i N_{i,k}(u), \text{ with knot set } (u_i)_{i=1}^{n+k}$$

to

$$\sum_{i=1}^{n+1} \mathbf{d}_i^1 N_{i,k}(u), \text{ with knot set } (u_i^1)_{i=1}^{n+1+k}.$$

where k is the order, n is the dimension, u is the parameter value, $\mathbf{d}_i$ is the i -th control point for the B-spline curve before knot insertion and the $\mathbf{d}_i^1$ is the i -th control point of the B-spline after knot insertion. The new knot set differs from the original one in having an additional knot inserted somewhere in the interval $[u_k, u_{n+1}]$.

If a knot $\hat{u}$ is to be inserted to the knot set coinciding with the knot u_{p+1} say, which already has multiplicity s , and the new knot set is denoted as

$$u_i^1 = \begin{cases} u_i & i \leq p \\ \hat{u} = u_{p+1} & i = p + 1 \\ u_{i-1} & i \geq p + 2, \end{cases} \quad (\text{E.1})$$

then the number of control point increases from n to $n + 1$, the knot set number increasing from $n + k$ to $n + k + 1$.

Since B-splines have a local modification property, only some of the control points will change. These new control points are obtained by linear interpolation of the previous control points according to the formula:

$$\mathbf{d}_i^1 = \alpha_i^1 \mathbf{d}_i + (1 - \alpha_i^1) \mathbf{d}_{i-1}$$

where $\alpha_i^1 = (\hat{u} - u_i^1)/(u_{i+k}^1 - u_i^1) = (\hat{u} - u_i)/(u_{i+k-1} - u_i)$ and $i = p - k + s + 2, p - k + s + 3, \dots, p$. More details of this algorithm can be found in [8]. By repeating the knot insertion algorithm many times the number of control points can be made to increase as required.

The knot insertion algorithm allows the user to add additional control points, i.e. design variables in the optimization, to the B-spline curve without changing the geometry. Therefore, it is possible for the user to start the optimiza-

tion iteration with a small number of control points, ending with a large number of design variable to describe the geometry.

The purpose in using the knot insertion algorithm is to decrease the number of optimization iterations whilst obtaining a better value for the objective function.

Application to a Poisson’s problem

We take the Poisson equation,

$$\nabla^2\Phi = -1.0 \quad in \quad \Omega$$

with the boundary condition

$$\Phi = 0 \quad on \quad \partial\Omega$$

and find the optimum shape maximising the maximum value in the domain (Ω) :

$$\max_{\partial\Omega} \max_{\Omega} \Phi,$$

subject to a geometric constraint, which is taken here to be the area of the domain, $A(\Omega)$ is a constant. After reorganizing, the objective function can be chosen as:

$$F = -\max_{\Omega} \Phi$$

Two cubic B-spline curves with nominal uniform knot sets are chosen to parameterize the geometry. After reorganization, the area of the whole domain is given as:

$$A(\Omega) = \sum_{i=1}^n (y_i \frac{u_{i+4} - u_i}{4}) - \sum_{j=1}^m (y_j \frac{u_{j+4} - u_j}{4})$$

where y_i, y_j are the control points in the upper curve, and lower curve respectively, n, m are the number of control points in the upper curve, and lower curve respectively, and u_i are the knots.

The gradient used in the optimization is computed using the finite difference method with the super-reduced method, given by the following equations:

$$\begin{aligned} \frac{\partial F}{\partial y_i} &\approx \frac{F(y_1, y_2, \dots, y_{i-1}, y_i + \epsilon, y_{i+1}, \dots, y_n + \delta) - F(y_1, y_2, y_3, \dots, y_n)}{\epsilon} \quad i = 1, 2, \dots, n-1 \\ \frac{\partial F}{\partial y_n} &= 0 \end{aligned}$$

where δ is given by solving the following constraints:

$$A(\Omega (y_1, y_2, \dots, y_{i-1}, y_i + \epsilon, y_{i+1}, \dots, y_n + \delta)) = k$$

and k is a constant.

Results

With the new method, the optimizer starts from the initial shape (see Figure 1), which has about 3300 triangles, and the number of control points for this initial geometry is 22. After 37 optimization iterations it finds the optimum shape (see Figure 2), which has about 13,000 number of triangles because mesh refinement has been employed. This optimum shape has 40 control points because the knot insertion algorithm has been incorporated.

The value of the objective function at the optimum geometry is -0.062333 .

The software used to solve the Poisson equation is that developed by Becker and Thompson [3].

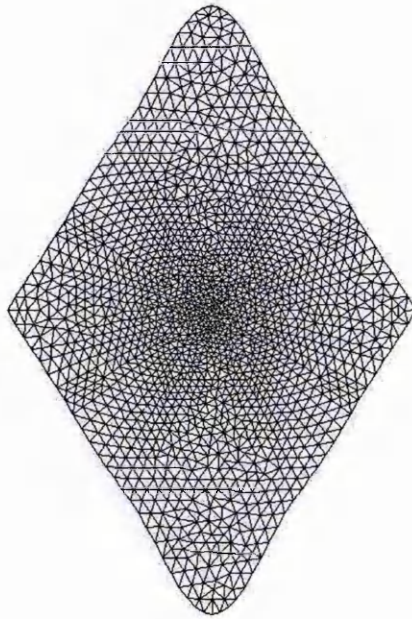


Figure 41: Initial shape

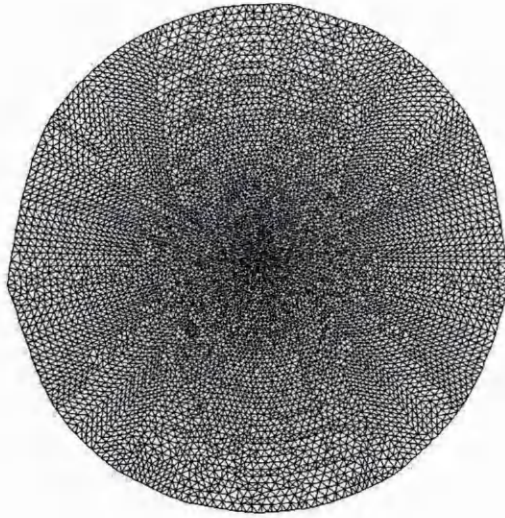


Figure 42: optimum shape produced with the new method

The BFGS method used in the paper has been added the following enhancements: super-reduction, the geometry model, an algorithm to deal with self-intersection, and mesh refinement in order to deal with the constrained shape optimization problem. When the BFGS method is used, we starts from the same initial shape and with the same mesh. The optimizer takes 44 iterations to find the optimum shape. In terms of number of iterations, the new methods decreases the iteration counts by about 20%. The performance comparison of these two methods are shown in Figure 3.

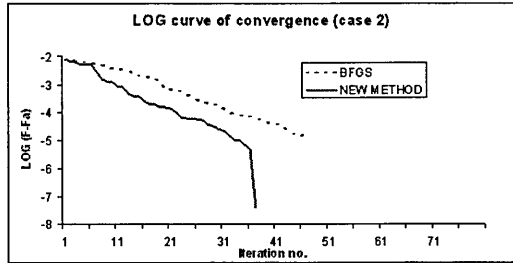


Figure 43: Convergence performance comparison with BFGS method

In the above graph, in iteration 7, the value of the objective function actually increases instead of decreasing because the shape is going to self-intersect, and the algorithm produces replacement. We discuss the algorithm to deal with self-intersecting shapes in [18].

The discretization error has also been investigated. Below is a table that reflects the influence of discretization error on the PDE solution:

number of nodes	error in solution	rate
814	0.0002161	
3169	5.318e-05	4.06

Table 22: Discretization error analysis for the solution of Poisson’s equation

The above table shows that when the number of triangles is increased to about 4 times, then the discretization error will decrease to about $1/4$, in agreement with theory.

Application for airfoil design

The purpose of this application is to find optimum shapes to minimize the drag of transonic airfoils while keeping the lift coefficient constant and keeping the maximum thickness of the airfoil constant. In this application, we follow Zhong and Qiao's work [20] using the B-spline method to parameterize the geometry. The shape modifications are parameterized by cubic B-splines with a nominal uniform knot set. The geometry of the airfoil is given by the following equation.

$$Y(\Gamma) = Y(\Gamma_{ini}) + \sum_{i=1}^n P_i N_i(X),$$

where $Y(\Gamma)$, $Y(\Gamma_{ini})$, represent the y coordinate of the design geometry and initial geometry respectively, P_i are the control points of the B-spline, and $N_i(X)$ are the B-spline basis functions.

The transonic airfoil analysis code used in this paper is the NPUFOIL program, see [20]. The program is based on a combination of the full potential equation solver and the boundary layer flow analysis. The flow and the aerodynamic characteristics are computed via viscous/inviscid iteration. In this design case, the symmetric airfoil NACA0012 is chosen as the initial geometry. The lift coefficient $C_L = 0.5$, Mach number $M_a = 0.77$, and the Reynolds number $Re = 6.5 \times 10^6$ are kept unchanged. For comparison, both the new method and the EXTREM [15] method are used in this design case. For the new method, the geometry is first parameterized by 5 free control points, and finally it reaches the optimum solution which has 9 free control

points and the drag is reduced from 0.0182 to 0.0070. The optimum solution of the control point is:

$$\begin{aligned}
 P_1 &= -0.571028 & P_2 &= -0.70975 \\
 P_3 &= -0.41511 & P_4 &= 0.13373 \\
 P_5 &= 0.78084 & P_6 &= 1.3277 \\
 P_7 &= 0.98959 & P_8 &= 0.56642 \\
 P_9 &= 0.13606
 \end{aligned}$$

The drag coefficient has been decreased by 62% (from 0.182 to 0.0070) for both methods, but the EXTREM method uses 408 function calls while the new method uses only 233. The CPU time per function call for these methods, (CPU/FC) is nearly the same. The performance comparison table for the two methods is given below:

Methods	Cd	iterations	FC	CPU/FC
EXTREM	0.0070	N/A	408	0.22246
New Method	0.0070	20	233	0.22418

Table 23: Performance comparison of the new method and the EXTREM method

Fig. 4 shows the comparison of pressure distribution in design case of NACA 0012, and Fig. 5 shows the comparison of the drag coefficient for initial and optimal geometry.

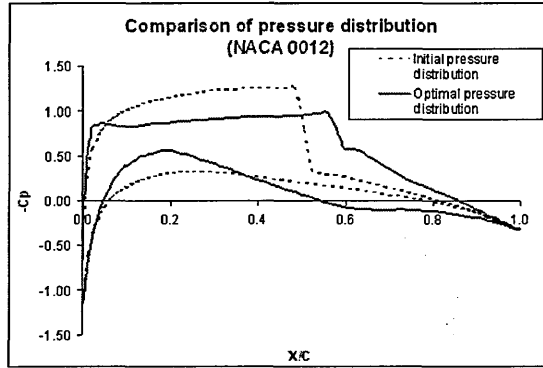


Figure 44: Initial and optimal pressure distribution for NACA 0012

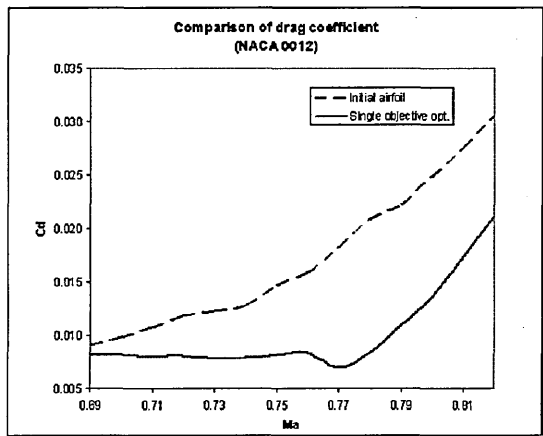


Figure 45: The comparison of drag coefficients for initial airfoil and optimal airfoil (NACA 0012)

Conclusions

In this paper, a new efficient method for shape optimization using B-spline shape functions combined with knot insertion algorithm and the super-reduced

method is given. This method has good performance for both mathematical problems and airfoil design applications. In the application to a Poisson's equation problem, the new method uses only 37 iterations while the BFGS method uses 44 iterations.

In the airfoil design case, the drag coefficient has been reduced significantly, by about 62% with only 233 function calls, saving about 42% CPU time compared to the EXTREM method. Using this technique researchers will be able to accelerate their optimization as long as B-spline shape functions are used to parameterize the geometry.

In the future we will extend the range of application to include 3-dimensional problems.

F An example of using adjoint method

An example of shape optimization problem is given here in order to explain the general procedure of a shape optimization problem without geometric constraints. This example can be found in [57] with small changes.

Suppose we have an optimization problem in 2-dimension. The objective function is:

$$F(\alpha) = \frac{1}{2} \int_{\Gamma} \left(\frac{\partial \Phi}{\partial x} - p^* \right)^2 dx \quad (\text{F.1})$$

where α is the design variable, p^* is a known distribution (desired distribution). The constraint is a PDE, stated below:

$$\nabla^2 \Phi = 0 \quad \text{in } \Omega \quad (\text{F.2})$$

$$\frac{\partial \Phi}{\partial \mathbf{n}} = \frac{\partial \alpha}{\partial x} \quad \text{on } \Gamma$$

$$\Phi = k \quad \text{on } \Gamma_0$$

where the domain Ω is a rectangle whose bottom boundary Γ has a control variable to be optimised, Γ_0 is the rest of boundary of Ω , and k is a constant.

Considering a small perturbation of the design variables by $\epsilon \tilde{\alpha}$, the corresponding change in Φ is given by $\epsilon \tilde{\Phi}$, which satisfies:

$$\nabla^2 \tilde{\Phi} = 0 \quad \text{in } \Omega \quad (\text{F.3})$$

$$\frac{\partial \tilde{\Phi}}{\partial \mathbf{n}} = \frac{\partial \tilde{\alpha}}{\partial x} \quad \text{on } \Gamma$$

$$\tilde{\Phi} = 0 \quad \text{on } \Gamma_0$$

The variation in the function is:

$$\begin{aligned} \delta F = F(\alpha + \epsilon \tilde{\alpha}) - F(\alpha) &= \frac{1}{2} \int_{\Gamma} \left[\left(\frac{\partial \Phi}{\partial x} + \epsilon \frac{\partial \tilde{\Phi}}{\partial x} - p^* \right)^2 - \left(\frac{\partial \Phi}{\partial x} - p^* \right)^2 \right] dx \\ &= \epsilon \int_{\Gamma} \left(\frac{\partial \Phi}{\partial x} - p^* \right) \frac{\partial \tilde{\Phi}}{\partial x} dx + O(\epsilon^2) \\ &= -\epsilon \int_{\Gamma} (\Phi_x - p^*)_x \tilde{\Phi} dx + O(\epsilon^2) \end{aligned} \quad (\text{F.4})$$

where integration by parts have been used in the last equality, We use Φ_x represent $\frac{\partial \Phi}{\partial x}$ for simplicity. In order to eliminate the dependency of δF on $\tilde{\Phi}$, an arbitrary function λ is chosen which is defined in the same domain. From equation 55, we have:

$$\begin{aligned}
0 = \int_{\Omega} \lambda \nabla^2 \tilde{\Phi} dA &= \int_{\Omega} \nabla \cdot (\lambda \nabla \tilde{\Phi}) dA - \int_{\Omega} \nabla \tilde{\Phi} \cdot \nabla \lambda \\
&= \int_{\partial\Omega} \lambda \frac{\partial \tilde{\Phi}}{\partial \mathbf{n}} ds + \int_{\Omega} \tilde{\Phi} \nabla^2 \lambda dA - \int_{\Omega} \nabla \cdot (\tilde{\Phi} \nabla \lambda) dA \\
&= \int_{\Omega} \tilde{\Phi} \nabla^2 \lambda dA + \int_{\partial\Omega} \left(\lambda \frac{\partial \tilde{\Phi}}{\partial \mathbf{n}} - \tilde{\Phi} \frac{\partial \lambda}{\partial \mathbf{n}} \right) ds \quad (\text{F.5})
\end{aligned}$$

Note: in the above equation, the divergence theorem, $\nabla(a \cdot b) = b \cdot \nabla a + a \cdot \nabla b$ and the definition of directional derivative are repeatedly used. Adding the right hand term of equation 57, multiplied by ϵ , to δF . We obtain:

$$\delta F = -\epsilon \int_{\Gamma} (\Phi_x - p^*)_x \tilde{\Phi} dx + \epsilon \int_{\Omega} \tilde{\Phi} \nabla^2 \lambda dA + \epsilon \int_{\partial\Omega} \left(\lambda \frac{\partial \tilde{\Phi}}{\partial \mathbf{n}} - \tilde{\Phi} \frac{\partial \lambda}{\partial \mathbf{n}} \right) ds + O(\epsilon^2)$$

After rearranging and using $\int_{\partial\Omega} = \int_{\Gamma} + \int_{\Gamma_0}$, we get:

$$\delta F \approx \epsilon \int_{\Omega} \tilde{\Phi} \nabla^2 \lambda dA - \epsilon \int_{\Gamma} \left[(\Phi_x - p^*)_x + \frac{\partial \lambda}{\partial \mathbf{n}} \right] \tilde{\Phi} ds - \epsilon \int_{\Gamma_0} \frac{\partial \lambda}{\partial \mathbf{n}} \tilde{\Phi} ds + \epsilon \int_{\partial\Omega} \lambda \frac{\partial \tilde{\Phi}}{\partial \mathbf{n}} ds \quad (\text{F.6})$$

Because λ can be any function, so a good choice of λ is:

$$\begin{aligned}
\nabla^2 \lambda &= 0 \quad \text{in } \Omega \\
(\Phi_x - p^*)_x + \frac{\partial \lambda}{\partial \mathbf{n}} &= 0 \quad \text{on } \Gamma \\
\lambda &= 0 \quad \text{on } \Gamma_0
\end{aligned} \quad (\text{F.7})$$

Obviously this choice can eliminate all of the dependence of δF on $\tilde{\Phi}$. Hence:

$$\delta F = \epsilon \int_{\Gamma} \lambda \frac{\partial \tilde{\Phi}}{\partial \mathbf{n}} dx + O(\epsilon^2) = \epsilon \int_{\Gamma} \lambda \tilde{\alpha}_x dx + O(\epsilon^2) = -\epsilon \int_{\Gamma} \tilde{\alpha} \lambda_x dx + O(\epsilon^2) \quad \text{(F.8)}$$

where the integration by parts is used in the last equality.

Up to now it is obvious to obtain the variation of the objective function caused by the known perturbation in design variables. The next stage discusses the geometry representation.

Suppose:

$$\alpha(x) = \sum_{j=1}^n \alpha_j f_j(x) \quad \text{(F.9)}$$

where $f_j(x)$ are functions, α_j are coefficients of the corresponding functions.

Considering a perturbation in α_j , then we get:

$$\delta F = -\epsilon \sum_{j=1}^n \tilde{\alpha}_j \int_{\Gamma} \lambda_x f_j(x) dx \quad \text{(F.10)}$$

and hence:

$$\frac{\partial F}{\partial \alpha_j} = - \int_{\Gamma} \lambda_x f_j(x) dx \quad \text{(F.11)}$$

This is all that we need to apply an optimization scheme.

ity. The R-tree based methods seem to be most robust for higher dimensions, provided that the fanout of the R-tree nodes remains > 2 . Experiments [21] indicate that R^* -trees work well for up to 20 dimensions. The feature extraction method should therefore be such that a few features are sufficient to differentiate between objects.

We propose to use the *Discrete Fourier Transform* [20] for feature extraction. Given a sequence, we transform it from the time domain to the frequency domain. We then index only on the first few frequencies, dropping all other frequencies. This approach addresses the two problems cited above as follows:

- *Completeness of feature extraction:* Parseval's theorem [20], discussed in Section 2, guarantees that the distance between two sequences in the frequency domain is the same as the distance between them in the time domain.
- *Dimensionality curse:* As we discuss in subsection 3.3, a large family of interesting sequences exhibit strong amplitudes for the first few frequencies. Using the first few frequencies then avoids the dimensionality problem, while still introducing few *false hits*. The false hits are removed in a post-processing step.

The organization of the rest of the paper is as follows. Section 2 gives some background material on the Discrete Fourier Transform, and introduces Parseval's theorem that provides the basis for the indexing technique we propose. A resume of our indexing technique is given in Section 3.

We also justify our choice of similarity measure and the selection of DFT for feature extraction in this section. Section 4 contains performance experiments that empirically show the effectiveness of our technique. We conclude with a summary in Section 5.

2 Discrete Fourier Transform

We start with a brief overview of the Discrete Fourier Transform (DFT). The importance of the DFT is the existence of a fast algorithm, the *Fast Fourier Transform (FFT)*, that can calculate the DFT coefficients in $O(n \log n)$ time. Further information on the Fourier transform can be found in any digital signal processing textbook, for example, [20].

The n -point *Discrete Fourier Transform* [20] of a signal $\vec{x} = [x_t]$, $t = 0, \dots, n-1$ is defined to be a sequence $\vec{X}$ of n complex numbers X_f , $f = 0, \dots, n-1$, given by

$$X_f = 1/\sqrt{n} \sum_{t=0}^{n-1} x_t \exp(-j2\pi ft/n) \quad f = 0, 1, \dots, n-1 \quad (1)$$

where j is the imaginary unit $j = \sqrt{-1}$. The signal $\vec{x}$ can be recovered by the inverse transform:

$$x_t = 1/\sqrt{n} \sum_{f=0}^{n-1} X_f \exp(j2\pi ft/n) \quad t = 0, 1, \dots, n-1 \quad (2)$$

X_f is a complex number (with the exception of X_0 , which is a real, if the signal $\vec{x}$ is real). There are some minor discrepancies among books: some define $X_f = 1/n \sum_{t=0}^{n-1} x_t \exp(-j2\pi ft/n)$... or $X_f = \sum_{t=0}^{n-1} x_t \exp(-j2\pi ft/n)$... We have followed the definition in (Eq 1), for it simplifies the

upcoming Parseval's theorem (Eq 4).

Definitions: For a complex number $c = a + jb = A \exp(j\phi)$

- $A \equiv |c|$ is said to be the *amplitude* and ϕ to be the *phase* of the number c .
- The *conjugate* c^* of c is defined as $a - jb$.
- The *energy* $E(c)$ of c is defined as the square of the amplitude ($E(c) \equiv |c|^2 \equiv c c^*$).
- The *energy* $E(\vec{x})$ of a sequence $\vec{x}$ is defined as the sum of energies at every point of the sequence:

$$E(\vec{x}) \equiv \|\vec{x}\|^2 \equiv \sum_{t=0}^{n-1} |x_t|^2 \quad (3)$$

A fundamental observation for this paper is Parseval's theorem [20]:

Theorem 1 (Parseval) Let $\vec{X}$ be the Discrete Fourier Transform of the sequence $\vec{x}$. Then we have

$$\sum_{t=0}^{n-1} |x_t|^2 = \sum_{f=0}^{n-1} |X_f|^2 \quad (4)$$

That is, the energy in the time domain is the same as the energy in the frequency domain.

The Discrete Fourier Transform inherits the following properties from the continuous Fourier transform. Let ' $\longleftrightarrow$ ' indicate Fourier pairs, i.e.,

$$[x_t] \longleftrightarrow [X_f] \quad (5)$$

means that $[X_f]$ is the Discrete Fourier Transform of $[x_t]$. The Discrete Fourier Transform is a *linear transformation*: If

$$[x_t] \longleftrightarrow [X_f]; [y_t] \longleftrightarrow [Y_f] \quad (6)$$

then

$$[x_t + y_t] \longleftrightarrow [X_f + Y_f] \quad (7)$$

$$[ax_t] \longleftrightarrow [aX_f] \quad (8)$$

Also, a shift in the time domain changes only the phase of the Fourier coefficients, but not the amplitude.

$$[x_{t-t_0}] \longleftrightarrow [X_f \exp(2\pi f t_0 j/n)] \quad (9)$$

Given the above, Parseval's theorem gives

$$\|\vec{x} - \vec{y}\|^2 \equiv \|\vec{X} - \vec{Y}\|^2 \quad (10)$$

The latter implies that the Euclidean distance between two signals $\vec{x}$ and $\vec{y}$ in the time domain is the same as their Euclidean distance in the frequency domain.

We believe that for a large number of time sequences of practical interest, there will be a few frequencies with high amplitudes. Thus, if we index only on the first few frequencies, we shall have few *false hits*. This is a *key observation* for our proposed method.

3 Proposed Technique

We propose using the square root of the *sum of squared differences* as the distance function between two sequences. Specifically, the distance $\mathcal{D}(\vec{x}, \vec{y})$ between two sequences $\vec{x}$ and $\vec{y}$ is the square root of the energy of the difference:

$$\mathcal{D}(\vec{x}, \vec{y}) \equiv \left(\sum_{t=0}^{n-1} |x_t - y_t|^2 \right)^{1/2} \equiv (E(\vec{x} - \vec{y}))^{1/2} \quad (11)$$

If this distance is below a user-defined threshold ϵ , we say that the two sequences are similar.

Efficient Similarity Search In Sequence Databases

Rakesh Agrawal

Christos Faloutsos*

Arun Swami

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract. We propose an indexing method for time sequences for processing similarity queries. We use the Discrete Fourier Transform (DFT) to map time sequences to the frequency domain, the crucial observation being that, for most sequences of practical interest, only the first few frequencies are strong. Another important observation is Parseval's theorem, which specifies that the Fourier transform preserves the Euclidean distance in the time or frequency domain. Having thus mapped sequences to a lower-dimensionality space by using only the first few Fourier coefficients, we use R^* -trees to index the sequences and efficiently answer similarity queries. We provide experimental results which show that our method is superior to search based on sequential scanning. Our experiments show that a few coefficients (1-3) are adequate to provide good performance. The performance gain of our method increases with the number and length of sequences.

*On sabbatical from the Dept. of Computer Science, University of Maryland, College Park. This research was partially funded by the Systems Research Center (SRC) at the University of Maryland, and by the National Science Foundation under Grant IRI-8958546 (PYI), with matching funds from EMPRESS Software Inc. and Thinking Machines Inc.

1 Introduction

Sequences constitute a large portion of data stored in computers. There have been several efforts to model time-sequenced data, to design languages to query such data, and to develop access structures to efficiently process such queries (see [25] for a bibliography). Most of the work, however, has focussed on "exact" queries. New emerging applications, particularly database mining applications [2], require that databases be enhanced with the capability to process "similarity" queries. The following are some examples of the similarity queries over sequence databases:

- Identify companies with similar pattern of growth.
- Determine products with similar selling patterns.
- Discover stocks with similar movement in stock prices.
- Find if a musical score is similar to one of the copyrighted scores.

Similarity queries can be classified into two categories:

- a. *Whole Matching.* The sequences to be compared have the same length

n.

- b. *Subsequence Matching.* The query sequence is smaller; we look for a subsequence in the large sequence that best matches the query sequence.

We concentrate on whole matching, and present an indexing technique that can be used to efficiently process such queries. Within the whole matching case, we consider the following problems:

- a1. *Range Query.* Given a query sequence, find sequences that are similar within distance ϵ .
- a2. *All-Pairs Query* (or 'spatial join'). Given N sequences, find the pairs of sequences that are within ϵ of each other.

The parameter ϵ is a distance parameter that controls when two sequences should be considered similar. It could be either user-defined, or determined automatically (eg., $\epsilon=10\%$ of the 'energy' of the query sequence; see Eq. 3 for the definition of 'energy').

Approximate matching has been attracting increasing interest lately. Motro described a user interface for vague queries [18]. Shasha and Wang [24] proposed an indexing method that uses the triangular inequality and some precomputed distances to prune the search. However, the space overhead of the method seems quadratic on the number of objects, which may make it prohibitive for large databases. Aurenhammer [5] surveyed recent research on Voronoi diagrams, along with their use for nearest neighbor queries. Although Voronoi diagrams work well for approximate matches in 2-dimensional spaces, they

need intricate transformations to work for a 3-d space, and they do not work at all for higher dimensionalities. Jagadish [15] suggested using a few minimum bounding rectangles to extract features from shapes and subsequently managing the resulting vectors using a spatial access method, like k-d-B-trees, grid files, etc.

For numerical sequences, we propose extracting k features from every sequence, mapping it to k -dimensional space, and then using a multidimensional index to store and search these points. The multidimensional indexing methods currently in use are R^* -trees [6] and the rest of the R -tree and k-d-Btree family [12, 14, 16]; linear quadrees [22]; and grid-files [19]. There are two subtle problems with this approach that must be addressed:

- *Completeness of feature extraction:* How to extract features, and how to guarantee that we do not miss any qualifying object (time sequence, in our case). To guarantee no "false dismissal", objects should be mapped to points in k -dimensional space such that the Euclidean distance in the k -dimensional space is *less than or equal* to the real distance between the two objects.
- *Dimensionality "curse":* Most multidimensional indexing methods scale exponentially for high dimensionalities, eventually reducing to sequential scanning. For linear quadrees, the effort is proportional to the hyper surface of the query region [13]; the hyper surface grows exponentially with the dimensionality. Grid files face similar problems, since they require a directory that grows exponentially with the dimensionality.

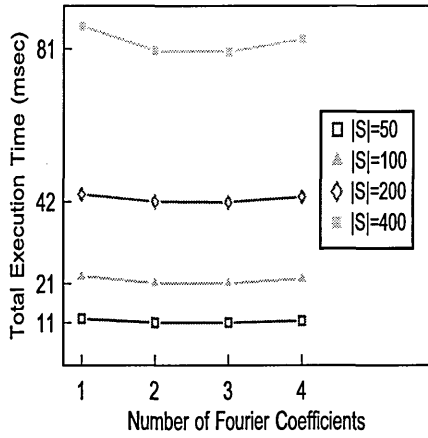


Figure 1: Time per query vs. # Fourier coefficients f_c , for range queries

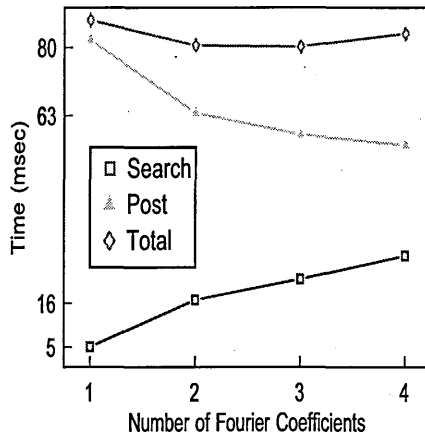


Figure 3: Breakup of the execution time, for range query ($|S| = 400$)

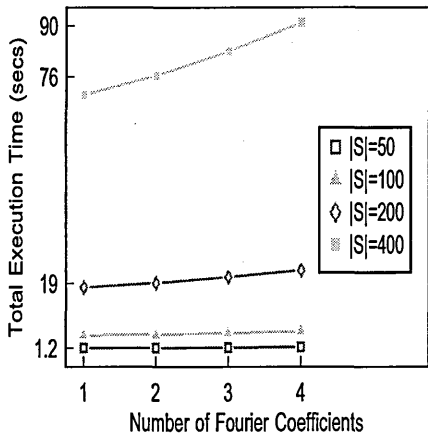


Figure 2: Time per query vs. # Fourier coefficients f_c , for all-pairs queries

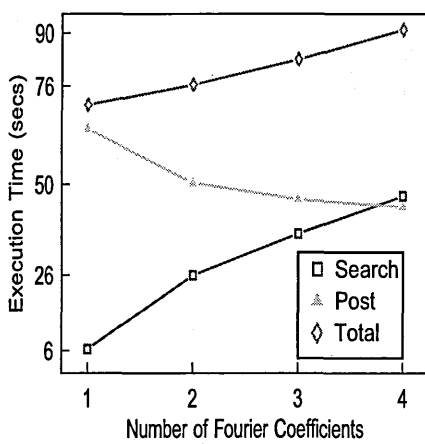


Figure 4: Breakup of the execution time, for all-pairs query ($|S| = 400$)

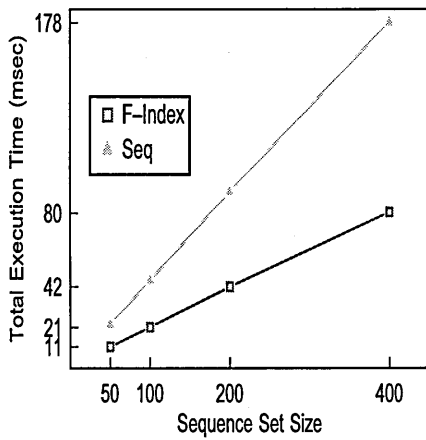


Figure 5: Time per query varying # sequences $|S|$, for range queries

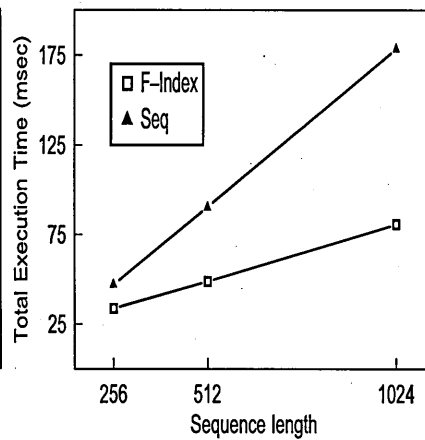


Figure 7: Time per query varying the sequence length n , for range queries

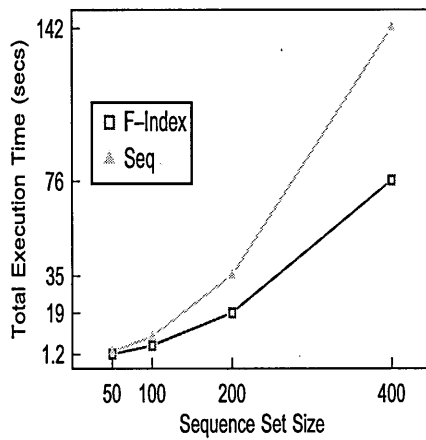


Figure 6: Time per query varying # sequences $|S|$, for all-pairs queries

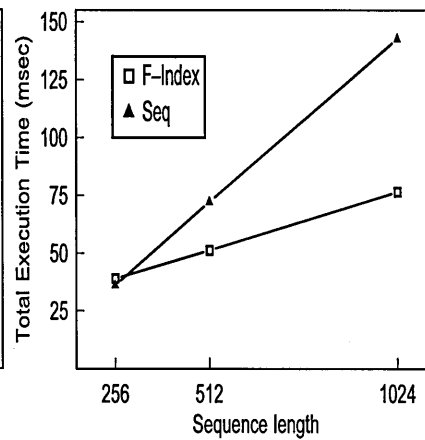


Figure 8: Time per query varying the sequence length n , for all-pairs queries

signals, called *black noise* [23]. Their energy spectrum follow $O(f^{-b})$, $b > 2$, which is even more skewed than the spectrum of the brown noise. Such signals model successfully, for example, the water level of rivers as they vary over time [17].

4 Performance Experiments

To determine the effectiveness of our proposed method (the *F-index* method), we compared it to a *sequential scanning* method. We used the *R**-tree for the index. For range queries, the sequential scanning method computes the distance between the query sequence and each data sequence. In our effort to do the best possible implementation for the sequential scanning, we stop the test as soon as the square of the distance exceeds ϵ^2 , and we declare the two sequences to be dissimilar. Thus, a data sequence is fully scanned only if it is similar to the query sequence. For ‘all-pairs’ queries, each sequence in the database is tested against every other sequence, for a total of $N(N-1)/2$ tests.

We investigated the following questions in these experiments:

- How to choose the number of Fourier coefficients to be retained (cut-off frequency f_c) in the *F-index* method. A larger f_c reduces the false hits but at the same time increases the dimensionality of the *R**-tree, and hence the search time.
- How does the search time grow as a function of number of sequences in the database?
- How does the length n of the sequences affect the performance?

4.1 Experimental setup

We generated synthetic sequences for the experiments. Each sequence $\vec{x} = [x_t]$ was a random walk:

$$x_t = x_{t-1} + z_t \quad (18)$$

where z_t ($t = 1, 2, \dots$) are independent, identically distributed (IID) random variables. For implementation convenience, each z_t variable is uniformly distributed in the range $(-500, 500)$. The probability distribution of each z_t is immaterial; the results would be the same had we chosen a gaussian distribution, or a fair, random coin. For each set $\mathcal{S}$ of N sequences, queries were generated by creating a distorted copy $[\tilde{x}_t]$ of each sequence $[x_t]$ in $\mathcal{S}$. This was accomplished by adding a small amount of noise to every x_t , i.e.,

$$\tilde{x}_t = x_t + p w_t \quad (19)$$

where $p=0.05$ and w_t ($t = 1, 2, \dots$) are IID random variables, each following a uniform distribution in the range $(-500, 500)$.

Let $\mathcal{Q}$ be the set of distorted sequences, which we shall use as queries. For range queries, we search $\mathcal{S}$ for sequences within distance ϵ for every distorted sequence in $\mathcal{Q}$. For all-pairs queries, we concatenate $\mathcal{S}$ and $\mathcal{Q}$, and ask for all sequence pairs within ϵ distance. The execution time for the *F-index* method includes both the search time in the *R**-tree and the post-processing time.

3.3 Using Few Fourier Coefficients for Indexing

Using a small value for the number of Fourier coefficients retained f_c does not affect the correctness — the F -index is a filter that returns a superset of the answer set. However, our proposed technique will not be very effective if the choice of a small f_c results in a large number of false hits.

The worst-case signal for our method is *white noise*, where each value x_t is completely independent of its neighbors x_{t-1} , x_{t+1} . The energy spectrum of white noise follows $O(f^0)$ [23], that is, it has the same energy in every frequency. This is bad for the F -index, because it implies that all the frequencies are equally important. However, we have strong reasons to believe that real signals have a skewed energy spectrum. For example, *random walks* (also known as *brown noise* or *brownian walks*) exhibit an energy spectrum of $O(f^{-2})$ [23], and therefore an amplitude spectrum of $O(f^{-1})$. Stock movements and exchange rates have been successfully modeled as random walks (e.g., [8, 17]). Using the data set available through ftp from *sfi.santafe.edu*, we show in [1] that the Fourier transform of the movement of the exchange rate between the Swiss franc and the US dollar follows closely the same $1/f$ behavior as for a random walk.

Our mathematical argument for keeping the first few Fourier coefficients agrees with the intuitive argument of the Dow Jones theory for stock price movement (see, for example, [9]). This theory tries to detect *primary* and *secondary* trends in the stock market movement, and ignores *minor* trends. Primary trends are defined as

changes that are larger than 20%, typically lasting more than a year; secondary trends show 1/3-2/3 relative change over primary trends, with a typical duration of a few months; minor trends last roughly a week. From the above definitions, we conclude that primary and secondary trends correspond to strong, low frequency signals while minor trends correspond to weak, high frequency signals. Thus, the primary and secondary trends are exactly the ones that our method will automatically choose for indexing.

In addition to stock movements and exchange rates, it is believed that several families of real signals are not white noise. For example, 2-d signals, like photographs, are far from white noise, exhibiting a few strong coefficients in the lower spatial frequencies. The JPEG image compression standard [26] exactly exploits this phenomenon, effectively ignoring the high-frequency components of the Discrete Cosine Transform, which is closely related to the Fourier transform. If the image consisted of white noise, no compression would be possible at all. Birkhoff's theory [23] claims that '*interesting*' signals, such as musical scores and other works of art, consist of *pink noise*, whose energy spectrum follows $O(f^{-1})$. The argument of the theory is that white noise with $O(f^0)$ energy spectrum is completely unpredictable, while brown noise with $O(f^{-2})$ energy spectrum is too predictable and therefore boring. The energy spectrum of pink noise lies in between. Signals with pink noise also have their energy concentrated in the first few frequencies (but not as few as in the random walk). In addition to the above, there is another group of

1-d and 2-d signals. In a recent survey for images (2-d signals), Brown [7](p. 367, sect. 4.2) mentions that one of the typical similarity measures is the cross-correlation (which reduces to the Euclidean distance, plus some additive and multiplicative constants). We have chosen the Euclidean distance, because (a) it is useful in many cases, as is (b) it can be used with *any* other type of similarity measure, as long as this measure can be expressed as the Euclidean distance between feature vectors in some feature space.

In fact, the Euclidean distance is the optimal distance measure for estimation [11], if signals are corrupted by Gaussian, additive noise. Thus, if $\vec{q}$ is our query and $\vec{x}$ is a corrupted version of it in the database, a searching method using the Euclidean distance should produce good results.

A valuable feature of the Euclidean distance is that it is preserved under orthonormal transforms. Other distance functions, like the L_p norms

$$L_p(\vec{x}, \vec{y}) = (\sum |x_i - y_i|^p)^{1/p} \quad (17)$$

do not have this property, unless $p = 2$ (because $L_2 \equiv$ Euclidean distance).

3.2 Using DFT

Having decided on the Euclidean distance as the distance measure, we would like a transform that (a) preserves the distance (b) is easy to compute and (c) concentrates the energy of the signal in few coefficients.

The distance-preservation requirement is met by any *orthonormal transform* [10], DFT being one of them. Orthonormal transforms form two classes: (1) the data-dependent ones, like the Karhunen-Loeve (K-L)

transform, which need all the data signals to determine the transformation matrix and (2) the data-independent ones, like the DFT, Discrete Cosine (DCT), Harr, or wavelet transform, where the transformation matrix is determined a-priori.

The data-dependent transforms can be fine-tuned to the specific data set, and therefore they can achieve better performance, concentrating the energy into fewer features in the feature vector. Their drawback is that, if the data set evolves over time, e.g., a recomputation of the transformation matrix may be required to avoid performance degradation, requiring expensive data reorganization. We, therefore, favor data-independent transforms.

Among them, we have chosen the DFT because it is the most well known, its code is readily available and it does a good job of concentrating the energy in the first few coefficients, as we shall see next. In addition, the DFT has the attractive property that the *amplitude* of the Fourier coefficients is invariant under shifts (Eq. 9). Thus, using Fourier transforms for feature extraction has the potential that our technique can be extended to finding similar sequences ignoring shifts.

Note that our approach can be applied with *any* orthonormal transform. In fact, our response time will improve with the ability of the transform to concentrate the energy: the fewer the coefficients that contain most of the energy, the faster our response time. Thus, the performance results presented next are just pessimistic bounds; better transforms will achieve even better response times.

The importance of Parseval's theorem (Eq 4) is that it allows to translate the query from the time domain to the frequency domain. Coupled with the conjecture that few Fourier coefficients are enough, it allows us to build an effective index with a low dimensionality.

The following is a resume of our proposed technique:

1. Obtain the coefficients of the Discrete Fourier Transforms of every sequence in the database.
2. Build a multidimensional index using the first f_c Fourier coefficients, where f_c stands for 'cut-off frequency'. Thus, each sequence becomes a point in a $2f_c$ -dimensional space (recall that the Fourier coefficients are complex numbers). We discuss in subsection 3.3 why f_c can be taken to be small (< 5). As discussed earlier, we recommend the R^* -trees as the indexing structure, since it has been shown to work well for at least up to 20 dimensions [21]. This index will be called ' F -index' henceforth.
3. For a range query, obtain the first f_c Fourier coefficients of the query sequence. Use the F -index to retrieve the set of matching sequences that are at most ϵ distance away from the query sequence.
4. For an all-pairs query, we do a spatial join using the F -index. The result of the join will be a superset of the answer set.
5. The actual answer set is obtained in a post-processing step in which the actual distance between two sequences is computed in the time domain and only those within ϵ distance are accepted.

The 'completeness' of this method is based on the following lemma:

Lemma 1 *The F -index introduces no false dismissals.*

We only give the proof for range queries; the proof for 'all-pairs' queries is very similar. Suppose we want all sequences $\vec{x}$ that are similar to a query sequence $\vec{q}$, within distance ϵ , i.e.:

$$\mathcal{D}(\vec{x}, \vec{q}) \leq \epsilon \quad (12)$$

or, equivalently:

$$\|\vec{x} - \vec{q}\|^2 = \sum_{t=0}^{n-1} |x_t - q_t|^2 \leq \epsilon^2 \quad (13)$$

Using Parseval's theorem (Eqs. 4, 10), we want all $\vec{X}$ such that

$$\|\vec{X} - \vec{Q}\|^2 = \sum_{f=0}^{n-1} |X_f - Q_f|^2 \leq \epsilon^2 \quad (14)$$

Keeping only the first $f_c < n$ coefficients, we have

$$\sum_{f=0}^{f_c-1} |X_f - Q_f|^2 \leq \sum_{f=0}^{n-1} |X_f - Q_f|^2 \leq \epsilon^2 \quad (15)$$

Thus, equation (14) implies the following condition

$$\sum_{f=0}^{f_c-1} |X_f - Q_f|^2 \leq \epsilon^2 \quad (16)$$

In other terms, the condition of (Eq. 16) will retrieve all $\vec{X}$ that are in the answer, plus some false hits. Thus, our index acts as a filter that returns a superset of the answer set.

■

3.1 Choice of Similarity Measure

The similarity measure is clearly application-dependent. Several similarity measures have been proposed, for

Parameter	Symbol	Values	Default value
# Fourier coefficients kept	f_c	1, 2, 3, 4	2
# sequences in $\mathcal{S}$	$ \mathcal{S} $	50, 100, 200, 400	400
Length of each sequence	n	256, 512, 1024, 2048	1024
Distance (tolerance)	ϵ	$\text{sqrt}(1000 \times n)$	

Table 1: Summary of experimental settings

We repeated each experiment 10 times by generating 10 sequence sets with different seeds, and averaged the execution times from these repetitions. Table 1 summarizes the parameters of the experiments.

4.2 Varying the cut-off frequency f_c

Figures 1 and 2 show the execution time per query for range and all-pairs queries respectively, for different number of sequences in $\mathcal{S}$. Figures 3 and 4 give the the total execution time by the F-index method for the two types of queries, broken into (a) search time in the R^* -tree and (b) post-processing time (where the ‘false hits’ are eliminated). The latter two graphs have been plotted for $|\mathcal{S}| = 400$ sequences in $\mathcal{S}$.

As the number of Fourier coefficients (‘cut-off frequency’ f_c) increases, the dimensionality of the R^* -tree increases. Recall that each Fourier coefficient, being a complex number, increases the dimensionality of the R^* -tree by 2. The increase in dimensionality results in better index selectivity, which gives fewer false hits. This reduction in false hits is reflected in the post-processing time, which decreases with the cut-off frequency. However, the time to search the R^* -tree increases with the dimensionality, because the fanout is smaller, and the tree is taller. Figures 3 and 4 are in complete agreement with the above intuitive arguments.

Given the trade-off between the tree-search time and the post-processing time, it is natural to expect that there is an ‘optimal’ f_c . Indeed, the total execution time of our method shows such a minimum, as illustrated in Figures 1 and 2. Notice that this minimum is rather flat, and, more importantly, it occurs for small values of the cut-off frequency f_c . This experiment confirms our early conjecture that we can effectively use a small number of Fourier coefficients for indexing sequences.

For the rest of the experiments, we kept $f_c=2$ Fourier coefficients for indexing, resulting in a 4-dimensional R^* -tree.

4.3 Varying the number of sequences in the database

The next experiment compares the F -index method with the sequential scanning method for increasing number of sequences in the database. Figures 5 and 6 show

the execution time per query for range and all-pairs queries respectively, for different values of the number of sequences in $|S|$. Clearly, the F -index method outperforms the sequential scanning. As the number of sequences increases, the gain of the F -index method increases, making this method even more attractive for large databases.

4.4 Varying the length of sequences

First, we show the results for range queries. We varied the length of sequences, keeping the number sequences in S fixed to 400. The distance parameter ϵ was set to $(1000 \times n)^{1/2}$ (where n is the length of a sequence). Figure 7 shows the execution time per query for range queries for different sequence lengths. The gain of the F -index method increases with n .

Figure 8 shows the results of the experiments for all-pairs queries. The trends are similar with the ones for range queries.

4.5 Discussion

The major conclusions from our experiments are:

- The minimum in the execution time for both range and ‘all-pairs’ queries is achieved for a small number of Fourier coefficients ($f_c = 1 - 3$). Moreover, the minimum is rather flat, which implies that a sub-optimal choice for f_c will give search time that is close to the minimum.
- Increasing the number of sequences in the database results in higher gains for our method.
- Increasing the length of the sequences n also results in higher gains for our method.

Thus, the experiments show that the proposed F -index method achieves increasingly better performance, as the volume of the data increases.

Finally, we should mention that we also examined whether a ‘naive’ feature extraction method would work as well. For example, consider a method that keeps the first few values of each time sequence, and indexes on them. We carried out an experiment in which we indexed on the first 10 values of each time sequence. The performance of this method was very poor compared to the F -index method; there were many false hits, resulting in a large post-processing time. Judging that further details are of little interest, we omit the experimental results.

5 Summary

We proposed a method to index time sequences for similarity searching. The major highlights of this method are:

- The use of an orthonormal transform, and specifically, the Discrete Fourier Transform, to extract features from a sequence. The attractive property of the DFT is that the Euclidean distance in the time domain is preserved in the frequency domain, thanks to Parseval's theorem. Thus, the DFT fulfills the "completeness of feature extraction" criterion. In addition, the DFT is fast to compute ($O(n \log n)$).
- The recognition that a large family of sequences have only a few (f_c) strong Fourier coefficients. For example, random walks, stock price movements, exchange rates, exhibit an amplitude spectrum of $O(1/f)$. Ignoring the weak coefficients, we introduce a few false hits, but no false dismissals. The importance of this observation is that it avoids the "dimensionality curse" at the expense of a modest post-processing cost. Keeping the first f_c coefficients, each sequence becomes a point in a $2f_c$ -dimensional space (recall that the Fourier coefficients are complex numbers).
- The use of spatial access methods, and specifically R^* -trees, to index those points. We believe that R^* -trees are more robust than their competitors, for medium dimensionalities.

Extensive empirical evaluation demonstrated the effectiveness of the proposed method. We generated random walks, which model well stock price movements. The conclusions from our experiments are the following: (a) the execution time of our method shows a rather flat minimum for a small cut-off frequency ($f_c \approx 1-3$) (b) compared to sequential scanning, our method achieves better gains with increasing number of sequences and increasing length. Thus, our method will be more and more attractive, as the volume of the database increases to Gigabytes and Terabytes.

Although we have made certain choices (Euclidean distance between sequences in time domain for similarity measure, DFT for feature extraction, and R^* tree for maintaining indexes), our technique can be trivially adapted for

- any similarity measure that can be expressed as the Euclidean distance between feature vectors in some feature space
- any distance-preserving (eg., orthonormal) transform (the more the energy concentrated on few coefficients, the faster our response time)
- any multi-dimensional index that performs well for the number of features used for indexing.

Future work could examine the following issues

- Examination of other orthonormal transformations, in addition to the Discrete Fourier Transform.
- Extensions of our approach to 2-d and higher-dimensionality signals (e.g., images), in addition to 1-d signals (time sequences) that we have examined.

The work reported in this paper has been done in the context of the Quest project [2] at the IBM Almaden Research Center. In Quest, we are exploring the

various aspects of the database mining problem. Besides the problem of queries over large sequences, some other problems that we have looked into include the enhancement of the database capability with the classification queries [3] and with “what goes together” kinds of association queries [4]. The eventual goal is to build an experimental system that can be used for mining rules embedded in massive databases. We believe that database mining is an important application area, combining commercial interest with intriguing theoretical questions.

Acknowledgements: We thank Myron Flickner for several constructive comments and for his help with Parseval’s theorem. We also thank Harpreet Sawhney for pointing out the optimality of the Euclidean distance as the similarity measure under Gaussian, additive noise.

References

- [1] R. Agrawal, C. Faloutsos, and A. Swami, “Efficient Similarity Search In Sequence Databases”, Research Report, IBM Almaden Research Center, San Jose, California, 1993.
- [2] R. Agrawal, T. Imielinski, and A. Swami, “Database Mining: A Performance Perspective”, *IEEE Transactions on Knowledge and Data Engineering*, Special issue on Learning and Discovery in Knowledge-Based Databases, (to appear).
- [3] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami, “An Interval Classifier for Database Mining Applications”, *VLDB 92*, Vancouver, August 1992.
- [4] R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules between Sets of Items in Large Databases”, *ACM SIGMOD*, Washington D.C., May 1993.
- [5] F. Aurenhammer, “Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure” *ACM Computing Surveys* 23(3):345-405, Sept. 1991.
- [6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, “The R*-tree: an efficient and robust access method for points and rectangles”, *ACM SIGMOD*, pages 322-331, May 1990.
- [7] L. G. Brown, “A Survey of Image Registration Techniques”, *ACM Computing Surveys*, 24(4), pages 325-376, December 1992.
- [8] C. Chatfield, *The Analysis of Time Series: an Introduction*, Chapman and Hall, London & New York, 1984, Third Edition.
- [9] R. D. Edwards and J. Magee, *Technical Analysis of Stock Trends*, John Magee, Springfield, Massachusetts, 1966, 5th Edition, second printing.

- [10] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1990, 2nd Edition.
- [11] A. Gelb, *Applied Optimal Estimation*, MIT Press, 1986.
- [12] A. Guttman, "R-trees: a dynamic index structure for spatial searching", *Proc. ACM SIGMOD*, pages 47-57, June 1984.
- [13] G. M. Hunter and K. Steiglitz, "Operations on images using quad trees", *IEEE Trans. on PAMI*, PAMI-1(2):145-153, April 1979.
- [14] H. V. Jagadish, "Spatial search with polyhedra", *Proc. Sixth IEEE Int'l Conf. on Data Engineering*, February 1990.
- [15] H. V. Jagadish, "A retrieval technique for similar shapes", *Proc. ACM SIGMOD Conf.*, pages 208-217, May 1991.
- [16] D. Lomet and B. Salzberg, "The Hb-Tree: a Multiattribute Indexing Method with Good Guaranteed Performance", *ACM TODS*, 15(4), pages 625-658, December 1990.
- [17] B. Mandelbrot. *Fractal Geometry of Nature*, W.H. Freeman, New York, 1977.
- [18] A. Motro, "VAGUE: A User Interface to Relational Databases that Permits Vague Queries," *ACM Trans. on Information Systems (TOIS)*, 6(3), pages 187-214, July 1988.
- [19] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The grid file: an adaptable, symmetric multikey file structure", *ACM TODS*, 9(1):38-71, March 1984.
- [20] A. V. Oppenheim and R. W. Schaffer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
- [21] M. Otterman, "Approximate Matching with High Dimensionality R-trees", M.Sc. scholarly paper, Dept. of Computer Science, Univ. of Maryland, College Park, MD, 1992.
- [22] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1989.
- [23] M. Schroeder, *Fractals, Chaos, Power Laws: Minutes From an Infinite Paradise*, W.H. Freeman and Company, New York, 1991.
- [24] D. Shasha and T-L. Wang, "New techniques for best-match retrieval", *ACM TOIS*, 8(2):140-158, April 1990.
- [25] R. Stam and R. Snodgrass, "A Bibliography on Temporal Databases", *IEEE Bulletin on Data Engineering*, 11(4), Dec. 1988.
- [26] G. K. Wallace "The JPEG Still Picture Compression Standard", *CACM*, 34(4):31-44, April 1991.