

## University of Southampton Research Repository ePrints Soton

Copyright © and Moral Rights for this thesis are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holders.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given e.g.

AUTHOR (year of submission) "Full thesis title", University of Southampton, name of the University School or Department, PhD Thesis, pagination

UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING AND THE ENVIRONMENT

**Physics- and Engineering  
Knowledge-Based Geometry Repair  
System for Robust Parametric CAD  
Geometries**

by

Dong Li

A thesis submitted in partial fulfillment for the  
degree of Doctor of Philosophy

August 2013



UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND THE ENVIRONMENT

Computational Engineering and Design

Doctor of Philosophy

PHYSICS- AND ENGINEERING KNOWLEDGE-BASED GEOMETRY REPAIR  
SYSTEM FOR ROBUST PARAMETRIC GEOMETRIES

by Dong Li

In modern multi-objective design optimisation, an effective geometry engine is becoming an essential tool and its performance has a significant impact on the entire process. Building a parametric geometry requires difficult compromises between the conflicting goals of robustness and flexibility.

The work presents a solution for improving the robustness of parametric geometry models by capturing and modelling relative engineering knowledge into a surrogate model, and deploying it automatically for the search of a more robust design alternative while keeping the original design intent. Design engineers are given the opportunity to choose from a list of optimised designs to balance the robustness of the geometry and the original design intent. The prototype system is firstly tested on a 2D intake design repair example and shows the potential to reduce the reliance on human design experts in the conceptual design phase and improve the stability of the optimisation cycle. It also helps speed up the design process by reducing the time and computational power that could be wasted on flawed geometries or frequent human interferences. A case-study of the proposed repair system based on the design and analysis of a three-dimensional parametric turbine blade model has been set up. An automatic analysis workflow is set up and the results are summarised for setting up a repair database based on surrogate training methods. Positive repair results have been achieved and an automatic repair cycle for the blade model is being set up and tested. The proposed physics and engineering knowledge based geometry repair system for robust parametric geometries proves an effective tool for ensuring automation robustness and design flexibility.





# Contents

<b>Acknowledgements</b>	<b>1</b>
<b>Nomenclature</b>	<b>3</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Motivation . . . . .	3
1.2 About the thesis . . . . .	5
<b>2 Literature Review: Design and Optimisation, Surrogate Modelling, Knowledge Representation and Geometry Repair</b>	<b>7</b>
2.1 Computer-aided-design . . . . .	7
2.1.1 Computer aided geometric generation . . . . .	7
2.1.2 Shape parameterisation . . . . .	8
2.1.3 CFD and downstream applications . . . . .	9
2.2 Design optimisation . . . . .	10
2.2.1 Linear and nonlinear programming . . . . .	11
2.2.2 Population-based algorithms . . . . .	11
2.2.3 Gradient-based algorithms . . . . .	13
2.2.4 Computational engineering design and optimisation framework . . . . .	13
2.3 Surrogate modelling . . . . .	14
2.3.1 Response surface methodology . . . . .	15
2.3.2 Radial basis function networks . . . . .	15
2.3.3 Kriging . . . . .	17
2.3.4 Support vector machine and support vector regression . . . . .	17
2.4 Sampling plans . . . . .	19
2.5 Knowledge-based systems . . . . .	19
2.6 Geometry repair . . . . .	20
<b>3 Methodology</b>	<b>23</b>
3.1 Knowledge representation . . . . .	23
3.2 Using support vector regression as the knowledge surrogate . . . . .	24
3.2.1 The basic idea . . . . .	25
3.2.2 Finding $\mathbf{w}$ . . . . .	26
3.2.3 Finding $\mu$ . . . . .	28
3.2.4 Kernels . . . . .	29
3.2.5 Computing $\epsilon$ using $\nu$ -SVR . . . . .	30
3.2.6 Tuning SVR parameters . . . . .	31
3.2.7 Implementation . . . . .	31

3.3	B-spline representation and derivatives . . . . .	32
3.4	Geometry repair . . . . .	33
3.5	Summary . . . . .	34
<b>4</b>	<b>Case Study: Geometry Repair in the Aero-engine Intake Design</b>	<b>37</b>
4.1	Overview of the intake design problem . . . . .	37
4.2	The intake model and its parameterisation . . . . .	38
4.3	Physics based knowledge of the intake design and its representation . . . .	39
4.3.1	Consideration on the vertical distance between the fuselage and intake . . . . .	40
4.3.2	Consideration on the curvature limit of the intake duct . . . . .	40
4.3.3	Interference with the rear pressure bulkhead . . . . .	41
4.4	Repair result . . . . .	42
4.4.1	A simple case . . . . .	42
4.4.2	Repair of a design candidate with multiple flaws . . . . .	43
4.4.3	Illustration of the repair path . . . . .	48
4.5	A comparison of the penalty prediction landscape and the repair result between different surrogate methods . . . . .	48
4.6	A graphical user interface . . . . .	64
4.7	Summary . . . . .	65
<b>5</b>	<b>Aero-engine Intake Design Case with Aerodynamic Properties Calculated and Incorporated in the Knowledge Base</b>	<b>67</b>
5.1	The incorporation of flow modelling and simulation . . . . .	67
5.1.1	Meshing process and the corresponding GAMBIT journal file . . . .	68
5.1.2	CFD simulation process and the corresponding FLUENT journal file . . . . .	69
5.2	Verification of the CFD process . . . . .	72
5.2.1	Effect of using smaller $y^+$ value . . . . .	75
5.2.2	Grid convergence . . . . .	79
5.2.3	Turbulence models . . . . .	82
5.3	Sampling plan, penalty table and surrogate modelling . . . . .	83
5.3.1	Sampling plan . . . . .	83
5.3.2	The penalty table . . . . .	83
5.3.3	Surrogates . . . . .	85
5.3.3.1	RBF surrogate, cross validation and visualisation . . . . .	85
5.3.3.2	SVR surrogate, tuning and visualisation . . . . .	86
5.3.4	Data validation . . . . .	88
5.4	Repair . . . . .	90
5.5	Surrogate update . . . . .	98
<b>6</b>	<b>3D Case Study: Repair of a Turbine Blade Geometry</b>	<b>101</b>
6.1	Overview of a turbine blade design problem . . . . .	101
6.2	An automated turbine blade design and analysis framework . . . . .	103
6.2.1	CAD model setup and parameterisation . . . . .	103
6.2.1.1	Geometry generator developed in NX Open . . . . .	105
6.2.2	Preprocessing . . . . .	106
6.2.3	Stress analysis . . . . .	107

6.2.3.1	Von Mises Stress and yield criterion . . . . .	107
6.2.3.2	Analysis in SC03 . . . . .	108
6.3	Statistics of modes of failure . . . . .	110
6.4	Knowledge base setup . . . . .	114
6.4.1	Data collection . . . . .	114
6.4.2	Geometry quality evaluation and penalty setup . . . . .	116
6.4.3	Surrogate training and tuning . . . . .	117
6.5	Repair with the knowledge base . . . . .	119
<b>7</b>	<b>Conclusions</b>	<b>129</b>
7.1	Summary . . . . .	129
7.2	Contributions . . . . .	130
7.3	Recommendations . . . . .	131
7.3.1	Recommended application of the work . . . . .	131
7.3.2	Recommended future research . . . . .	131
7.3.2.1	Identifying potential critical designs automatically . . . . .	131
7.3.2.2	Comparison of other SVR techniques . . . . .	133
7.3.3	Recommendations on research methodology . . . . .	134
<b>A</b>	<b>Important Codes Used in the Thesis</b>	<b>135</b>
A.1	Geometry modelling . . . . .	135
A.1.1	model.m . . . . .	135
A.1.2	drawaircraft.m . . . . .	136
A.1.3	duct.m . . . . .	137
A.1.4	ductoffset.m . . . . .	138
A.2	Penalty functions . . . . .	139
A.2.1	penalty.m . . . . .	139
A.2.2	penalty1.m (abridged) . . . . .	139
A.2.3	penalty2.m (abridged) . . . . .	140
A.2.4	penalty3.m (abridged) . . . . .	140
A.3	Support vector regression and prediction . . . . .	141
A.3.1	SVR regression function.m . . . . .	141
A.3.2	SVR prediction function . . . . .	142
A.4	ES optimisation repair function . . . . .	143
A.5	MATLAB codes as described in Figure 5.1 . . . . .	144
A.6	Sample GAMBIT journal file . . . . .	150
A.7	Sample FLUENT journal file . . . . .	151
A.8	Turbine blade model repair automation . . . . .	153
A.8.1	Sample blade design variable file . . . . .	153
A.8.2	NX Open C file that reads blade design variables . . . . .	153
A.8.3	NX Open C file that creates a turbine blade 3D section . . . . .	157
A.8.4	Batch file that automates geometry generation . . . . .	173
A.8.5	Batch file that automates IGES format geometry generation . . . . .	173
A.8.6	Batch file that automates pm format geometry generation . . . . .	174
A.8.7	Batch file that automatically starts SC03 and loads SC03 executable file . . . . .	174
A.8.8	SC03 executable file for setup, meshing, analysis and output . . . . .	174

A.8.9	SC03 material property file . . . . .	176
A.8.10	3D evolutionary strategy optimisation MATLAB code for propos- ing turbine blade repair alternative . . . . .	177
A.8.11	MATLAB code for stress analysis . . . . .	180
<b>References</b>		<b>185</b>

# List of Figures

2.1	Illustration of hyperplanes in a 2-dimensional space . . . . .	18
3.1	Knowledge representation . . . . .	25
3.2	Methodology flowchart for setting up the repair system and the repair process . . . . .	35
4.1	A simplified 2D intake model . . . . .	39
4.2	Failed design example due to high curvature . . . . .	41
4.3	A faulty geometry due to structure interference . . . . .	42
4.4	Initial global search result with its Pareto front being marked by circle, and the initial choice being marked by a bold circle . . . . .	43
4.5	Suggested repair alternative of the original design, see Figure 4.3 for comparison . . . . .	44
4.6	A design candidate with multiple flaws . . . . .	45
4.7	Illustration of a repair path . . . . .	49
4.8	RBF with thin plate base prediction landscape . . . . .	50
4.9	RBF with Gaussian base prediction landscape . . . . .	51
4.10	RBF with inverse multi-quadric base prediction landscape . . . . .	51
4.11	SVR prediction landscape . . . . .	52
4.12	Analytical penalty function landscape . . . . .	53
4.13	Repair Pareto front for design 1: Predicted Penalty versus Minimum distance . . . . .	54
4.14	True penalties of the repair suggestions on the Repair Pareto front for design 1 . . . . .	54
4.15	Repair Pareto front for design 2: Predicted Penalty versus Minimum distance . . . . .	55
4.16	True penalties of the repair suggestions on the Repair Pareto front for design 2 . . . . .	55
4.17	Repair Pareto front for design 3: Predicted Penalty versus Minimum distance . . . . .	56
4.18	True penalties of the repair suggestions on the Repair Pareto front for design 3 . . . . .	56
4.19	Repair Pareto front for design 4: Predicted Penalty versus Minimum distance . . . . .	57
4.20	True penalties of the repair suggestions on the Repair Pareto front for design 4 . . . . .	57
4.21	Repair Pareto front for design 5: Predicted Penalty versus Minimum distance . . . . .	58

4.22	True penalties of the repair suggestions on the Repair Pareto front for design 5 . . . . .	58
4.23	Repair Pareto front for design 6: Predicted Penalty versus Minimum distance . . . . .	59
4.24	True penalties of the repair suggestions on the Repair Pareto front for design 6 . . . . .	59
4.25	Repair Pareto front for design 7: Predicted Penalty versus Minimum distance . . . . .	60
4.26	True penalties of the repair suggestions on the Repair Pareto front for design 7 . . . . .	60
4.27	Repair Pareto front for design 8: Predicted Penalty versus Minimum distance . . . . .	61
4.28	True penalties of the repair suggestions on the Repair Pareto front for design 8 . . . . .	61
4.29	Repair Pareto front for design 9: Predicted Penalty versus Minimum distance . . . . .	62
4.30	True penalties of the repair suggestions on the Repair Pareto front for design 9 . . . . .	62
4.31	Repair Pareto front for design 10: Predicted Penalty versus Minimum distance . . . . .	63
4.32	True penalties of the repair suggestions on the Repair Pareto front for design 10 . . . . .	63
4.33	A MATLAB GUI for generating and examining repair alternatives . . . .	65
5.1	Determination of aerodynamic properties . . . . .	68
5.2	An example of successful meshing . . . . .	70
5.3	An example of meshing failure . . . . .	71
5.4	A typical distribution of the total pressure in the intake . . . . .	71
5.5	A typical distribution of the total pressure at the engine face . . . . .	72
5.6	Residual convergence history . . . . .	73
5.7	Velocity streamlines . . . . .	74
5.8	Total pressure distribution profiles at the outlet of the intake ducts . . . .	75
5.9	Geometry 148 . . . . .	76
5.10	$y^+$ distribution along the duct walls for Geometry 147 . . . . .	77
5.11	$y^+$ distribution along the duct walls for Geometry 148 . . . . .	78
5.12	Goodness of fit plot of the Penalty 4 . . . . .	79
5.13	Goodness of fit plot of the penalty sum . . . . .	80
5.14	Comparison of pressure distribution with different grid scheme . . . . .	81
5.15	RBF surrogate landscape . . . . .	87
5.16	SVR surrogate landscape ( $\sigma = 0.4$ ) . . . . .	88
5.17	SVR surrogate landscape ( $\sigma = 0.5$ ) . . . . .	89
5.18	A design candidate with multiple flaws . . . . .	91
5.19	Illustration of the surrogate model update process . . . . .	98
6.1	A turbine blade with cooling holes for film cooling . . . . .	102
6.2	Turbine Blade cross-section view with illustrations of the design variables	104
6.3	Turbine blade shaded view . . . . .	104
6.4	Possible blade cooling hole position contours . . . . .	105

6.5	Turbine blade wireframe view . . . . .	106
6.6	Blade model imported into SC03 . . . . .	109
6.7	Blade model meshing . . . . .	110
6.8	Blade temperature contours . . . . .	111
6.9	Von-Mises stress contour map . . . . .	112
6.10	Von-Mises stress contour map with predicted blade deflection displayed . . . . .	113
6.11	Data collection process . . . . .	115
6.12	Top view of the original design $\mathbf{x}_o$ in NX . . . . .	119
6.13	Trimetric view of the original design $\mathbf{x}_o$ in NX . . . . .	120
6.14	Initial selection of the repair alternative after a global search . . . . .	121
6.15	Selection of the repair alternative in the hyper-circle . . . . .	122
6.16	Repaired blade model $\mathbf{x}_2$ displayed and meshed in SC03 . . . . .	123
6.17	Selection of the repair alternative in the hyper-circle in subsequent optimisation . . . . .	124
6.18	Suggested repair alternative $\mathbf{x}_r$ (same as $\mathbf{x}_3$ ) displayed and successfully meshed in SC03, with the predicted penalty equaling to 1.5413 . . . . .	125
6.19	Final repair alternative $\mathbf{x}_r$ analysed in SC03, showing deflected shape under constraints and thermal stress . . . . .	126
6.20	Repaired blade model $\mathbf{x}_2$ analysed in SC03, showing stress contours only . . . . .	127
7.1	$\alpha_{pm}$ histogram (10 bins) . . . . .	132
7.2	$\alpha_{pm}$ histogram (100 bins) . . . . .	133





# List of Tables

4.1	List of the horizontal positions, acceptable vertical position ranges and corresponding design variables for each control point . . . . .	39
4.2	Design alternatives based on different feasibility threshold levels . . . . .	45
4.3	Variables of ten faulty design cases . . . . .	50
4.4	Quality comparison of the surrogates . . . . .	64
5.1	Standard deviation obtained using different $y^+$ meshing schemes . . . . .	77
5.2	Variation of simulation functionals on multiple grid levels . . . . .	82
5.3	GCI and asymptotic rate for simulation functionals . . . . .	82
5.4	Sampling plan . . . . .	83
5.5	Penalties and statistics . . . . .	84
5.6	Normalised penalty value table . . . . .	85
5.7	RBF cross validation result . . . . .	86
5.8	$\sigma$ vs. Prediction Errors . . . . .	88
5.9	20 new designs, their true penalties, RBF and SVR predictions . . . . .	90
5.10	Comparison of prediction errors for different surrogates . . . . .	90
5.11	Predicted penalty values . . . . .	91
5.12	Design alternatives based on the RBF surrogate . . . . .	92
5.13	Repair suggestions based on the SVR surrogate ( $\sigma = 0.4$ ) . . . . .	94
5.14	Repair suggestions based on the SVR surrogate ( $\sigma = 0.5$ ) . . . . .	96
5.15	Update training points' true penalties and SVR predictions . . . . .	99
5.16	Comparison of surrogates between updates . . . . .	99
6.1	Blade variable definitions . . . . .	103
6.2	Error type and statistics . . . . .	114
6.3	Blade working-condition measurements . . . . .	116
6.4	Measurement extremes . . . . .	116
6.5	Penalty metric . . . . .	117
6.6	Penalty statistics . . . . .	117
6.7	Normalised penalty table and penalty sum . . . . .	118
6.8	Goodness of fit of SVR surrogate model trained by using different sizes of training data . . . . .	118
6.9	Original design variables and repair alternatives . . . . .	124



# Listings

A.1	model.m . . . . .	135
A.2	drawaircraft.m . . . . .	136
A.3	duct.m . . . . .	137
A.4	ductoffset.m . . . . .	138
A.5	penalty.m . . . . .	139
A.6	penalty1.m . . . . .	139
A.7	penalty2.m . . . . .	140
A.8	penalty3.m . . . . .	141
A.9	SVR regression function . . . . .	141
A.10	SVR prediction function.m . . . . .	142
A.11	ES optimisation repair function . . . . .	143
A.12	MATLAB code as described in Figure 5.1 . . . . .	144
A.13	Sample GAMBIT journal file . . . . .	150
A.14	Sample FLUENT journal file . . . . .	151
A.15	Sample blade design variable file . . . . .	153
A.16	NX Open C file that reads blade design variables . . . . .	154
A.17	NX Open C file that creates a turbine blade 3D section . . . . .	157
A.18	Batch file that automates geometry generation . . . . .	173
A.19	Batch file that automates IGES format geometry generation . . . . .	174
A.20	Batch file that automates pm format geometry generation . . . . .	174
A.21	Batch file that automatically starts SC03 and loads SC03 executable file .	174
A.22	SC03 executable file . . . . .	174
A.23	SC03 material property file . . . . .	176
A.24	3D evolutionary strategy optimisation MATLAB code for proposing tur- bine blade repair alternative . . . . .	177
A.25	MATLAB code for stress analysis . . . . .	180



# Declaration of Authorship

I, Dong Li, declare that the thesis entitled “Physics- and Engineering Knowledge-Based Geometry Repair System for Robust Parametric Geometries” and the work presented in it are my own. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

**This thesis is based, in part, on the following publications:**

- **D. Li**, A. Sóbester, A.J. Keane, (2010) A knowledge-based geometry repair system for robust parametric CAD models. In 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Orlando, Florida, 04 - 07 Jan 2010. 17pp, 1-17.
- **D. Li**, A. Sóbester, A.J. Keane, (2011) Physics- and engineering knowledge-based repair of parametric geometries, accepted for publication in AIAA Journal.



## Acknowledgements

First of all I would like to thank Professor Andy Keane and Dr. András Sóbester for their great supervision during my study. Their professional experience and great personality has greatly influenced me and led me onto the right track of research. Professor Andy J.Keane has very broad knowledge in the field of computational engineering and design. His great insight in research has inspired me academically to a very large extent. Dr. András Sóbester has always been supportive, encouraging and energetic. He is instrumental in giving me practical advices and instilling confidence. When I had a family crisis in 2010, the supervisors allowed me to suspend my study. When I returned, they offered me tremendous academic and spiritual support and patience without which I could not continue my study.

The working environment of the research group and University Technology Centre also helps to a great extent with my progress. I would like to thank Professor Jim Scanlan for his encouragement and constructive advice on my transfer thesis and final thesis. I also receive support from many researchers and my peer PhD students in and around the UTC office. David Toal, Felix Stanley, James Wong, Surya Mohan, Nikita Thakur, Ying Jiang, Alan Wong, Keven Briggs, Moreshe Wankhede, Sanjay Pant and Joshua Jeelson openly and generously share knowledge and experiences with me. I would like to thank all of my friends for your help. You are my precious memory in Southampton.

I have been financially supported by Engineering and Physical Sciences Research Council and Rolls-Royce plc under Dorothy Hodgkin Postgraduate Awards (DHPA) scheme. The scheme sponsors developing world student to come and study for PhD in top rated UK research facilities. I am very grateful to EPSRC and RR's sponsorship which have made my study in the University of Southampton possible.

I would like to pay special thanks to my parents and grandma for their spiritual support from home. My girlfriend Kangjing provides me with her greatest support and patience everyday. I would not have completed the thesis without their love.





# Chapter 1

## Introduction

### 1.1 Motivation

Computer-Aided Design (CAD) refers to the application of computer technology that assists engineers and designers in designing physical products ranging from aircraft, ships and cars to electronics and buildings. The applications include computer-based product specification, visualisation, analysis and optimisation, which have gradually become inseparable elements in the aerospace, automotive and other industries that produce complex, high-performance products over the past 30 years. CAD has greatly shortened the design cycle and reduced the design cost by providing a virtual laboratory for any product design. Throughout the rest of this thesis, an engineering design problem implies a computer-aided engineering design problem if not otherwise specified.

Complex engineering design problems usually start with a conceptual design phase. In this initial phase, the objective is to find the right combination of product parameters that satisfies all the design constraints and relevant regulations, while at the same time, optimise figures of merit, for example, cost, weight and aerodynamic drag. Essentially, this conceptual design phase is a highly global exploration of the possible design space. The exploration could be carried out in a design optimisation framework.

The practice of engineering design optimisation has gradually evolved from a manual, time consuming, and step-by-step approach to an automated optimisation process [Roy et al. (2008)]. The automated frameworks that have been developed are abundant in the literature and diverse in nature depending on the problem at hand. Nevertheless, most of them make use of a common component: a parametric geometry model, which is entirely defined by a set of design variables. Such models serve as the starting point for subsequent analysis and evaluation, such as computational fluid dynamics (CFD) or computational structural mechanics (CSM), the analysis outcome(s) being sent back into the optimisation framework, creating a design-evaluate-redesign workflow. In an

ideal workflow, the geometry engine is able to deliver a geometry model defined by the set of design variables as and when required by the optimiser.

Because of the highly global nature of conceptual design search, the geometry engine in the optimisation framework should be able to deliver a variety of different geometries defined by a wide range of design variable configurations without difficulty, i.e. the geometry engine should be flexible as well as robust. However, although parameterisation technology has been a research focus for at least 20 years and various parameterisation technologies having been developed (see [Samareh (2001)], for example), the control of the trade-off between the desire for robustness and the need for flexibility is still a pressing challenge of parametric geometry generation. An expedient measure for ensuring robustness is to place tight bound limits to design variables so that any combination of the variables in the trimmed design space leads to a feasible design. However, for complex geometry models, the infeasible regions often exhibit irregular shapes, and are therefore hard to avoid. The above measure will either lead to very limited design space that could be explored, or some remaining infeasible region(s), which will make the model generation process fail from time to time.

Up to now, there is no satisfactory solution to the above problem. Especially for general-purpose commercial CAD package, a flawless coverage of the design space is very difficult to realise. As a result, bespoke in-house geometry engines still dominate in the conceptual design phase. These bespoke engines are made according to the specific needs of an individual customer or product. They are usually more time-consuming, difficult and costly to set up and use than ready-to-use commercial CAD tools [Keane and Nair (2005a)]. Their applications are usually limited for specific problems, and within a company or an organisation. Furthermore, there is no mechanism to guarantee that the precious engineering experience and knowledge which is used in the construction of a bespoke geometry engine can be preserved and further reused. So far, there is relatively little work that has addressed the problem.

In this work, an automatic geometry repair system is proposed to handle the above problem. The system aims to repair geometrically or physically flawed geometries based on an engineering knowledge base, and assist the geometry engine to generate robust models without limiting its flexibility. The system would reduce the reliance on human design experts in the conceptual design phase and improve the stability of the optimisation cycle. It also helps speed up the design process by reducing the time and computational power that could be wasted on flawed geometries or frequent human interventions. The prototype system aims to provide the following capabilities: capturing and storing the knowledge of a design engineer; synthesising the knowledge into a general knowledge base; deploying the knowledge automatically to recommend a repaired geometry alternative as and when required; producing inferences that the human expert may not be able to devise in a reasonable amount of time. In a nutshell, the system could be a valuable tool in a fully or partially automated design optimisation cycle.

It should be noted that the automated design optimisation process could be hindered by some other problems, for example, some geometric features or topological errors that could not be automatically dealt with by CFD or CSM grid generation algorithms. This type of error is not strictly related to the parameterisation of the geometry and often unrelated to the engineer's domain knowledge. Therefore, they lie outside of the repair capability of the prototype system that is proposed in this thesis. In the literature, the practice of preparing geometries for subsequent manipulation such as meshing is usually referred to as "geometry clean-up" [Hu et al. (2002)], "de-featuring" [Gopalakrishnan and Suresh (2007)] or "healing" [Chong et al. (2007)]. These operations generally aim at improving the geometry's suitability for downstream applications. In this thesis, the term "geometry repair" refers to the operation that improves the overall feasibility of an unfeasible geometry by repairing its geometries parameter sets, rather than directly patching the geometry.

## 1.2 About the thesis

In Chapter 2, literature review relative to this research is given. Basic building blocks of design and optimisation in the field of aerospace engineering are first reviewed. These include shape parameterisation, computational fluid dynamics and downstream applications and design optimisations. Various design optimisation methods are reviewed. There include linear and nonlinear programming, various evolutionary algorithms and optimisation methods that require derivative information of the objective function. Existing surrogate modelling techniques are reviewed and individually examined. These include response surface methodology, radial basis functions, Kriging and support vector regression. Literature on incorporating prior knowledge into surrogate models are reviewed. In Chapter 3, the proposed methodology and algorithm of geometry repair is provided. Firstly a knowledge representation scheme is proposed. The support vector regression is used as a main tool to construct the knowledge base. Basic ideas, detailed mathematics and implementation of support vector regression are examined. A brief review to B-splines representations and their derivatives is given in Section 3.3, as case studies in Chapter 4–6 use B-splines to construct geometries, . The geometry repair idea is presented in Section 3.4.

In Chapter 4, the geometry repair method is applied to and tested on an aero inlet design case. This chapter overviews the inlet design problem first. Then the intake model and its parameterisation scheme is set up before demonstrating how to set up a knowledge base based on practical engineering knowledge and design consideration. Repair results are positive and are presented in Section 4.4. A comparison of the penalty prediction landscape and the repair result between different surrogate methods is given in Section 4.5. A MATLAB graphical user interface has been developed for the real-time

display of the repair result and an interaction interface with the engineer who makes the decision is presented in Section 4.6.

Chapter 5 adds computational fluid dynamics simulation results to the engine inlet design case study. In this chapter, the process for setting up the flow modelling and simulation is described. Knowledge base is updated and repair results are presented. The effect of adding more training points to an existing surrogate model is also investigated in this chapter. The last section of Chapter 5 discusses a few technical aspects in the setting up of the CFD process, i.e., effect of using smaller  $y^+$  value, grid convergence and the consideration of other turbulence models.

Chapter 6 extends the proposed methodology to a realistic 3D blade design case. After a brief review of a turbine blade design problem, the author set up an automated turbine blade design and analysis framework. Description of CAD model setup and parameterisation is followed by model preprocessing and stress analysis. Knowledge gathered from these steps are analysed and used for setting up a knowledge base. The repair of infeasible design is presented in Section 6.5.

Finally, conclusions and a few pointers for future research are presented in Chapter 7.

## Chapter 2

# Literature Review: Design and Optimisation, Surrogate Modelling, Knowledge Representation and Geometry Repair

This chapter reviews all essential aspects that are relevant to the proposed geometry repair system. A study on state-of-the-art practise in engineering design and optimisation is presented in Section 2.1, placing the geometry repair in the context of the engineering design process. In Section 2.3 surrogate modelling techniques are reviewed. The next section reviews how the relevant engineering knowledge is represented, captured and reused. Section 2.6 reviews other literature which deals with the problem of geometry repair.

## 2.1 Computer-aided-design

### 2.1.1 Computer aided geometric generation

The Sketchpad [Sutherland (1963)] is widely acknowledged as the first modern Computer Aided Geometric Design(CAGD) program and the ancestor to many CAD systems. The earliest CAGD systems first appeared in industry during the 1960s and 70s, mainly in large aerospace and automotive companies. They served as the replacement of the traditional “pencil-and-eraser” approach of 2D drawing practice, eliminating the need for drafting departments. Major technological breakthroughs were made in the 1980s as the

first 3D solid modelling and feature-based parametric modelling were introduced [Shah and Mantyla (1995)]. These days, CAGD is much more than a drawing or drafting aid. It is used for the specification and visualisation of detailed representation of 3D models and 2D drawing, as well as an integrated component in product lifecycle management. The aerospace industry is dominated by commercial CAD software including CATIA (Computer Aided Three-dimensional Interactive Application) from Dassault Systemes<sup>1</sup>, NX (originally named Unigraphics) from Siemens<sup>2</sup> and Pro/ENGINEER from the Parametric Technology Corporation<sup>3</sup>. Open source industrial alternatives to proprietary 3D modelling software such as Open CASCADE<sup>4</sup> are also available. Their source code can be freely adapted, modified and enriched according to particular applications. The absence of licence fee of open source modelling software can reduce costs of projects.

### 2.1.2 Shape parameterisation

The purpose of shape parameterisation is to provide an efficient, systematic method for delivering different design definitions and geometry manipulations. The way in which the geometry is parameterized is especially important in the process of design optimisation and is a key consideration for the setup of geometry engines because it is generally desired that the parameterisation is both flexible and robust for the geometry in question. A flexible parameterisation would allow more geometries be generated, thus allow a larger design space be explored in the design optimisation. A robust parameterisation would reduce the possibility that a failing geometries to be generated so that design optimisation can be carried out more smoothly.

The NACA series of airfoils is an early and successful parameterisation attempt in designing aircraft airfoils [Jacobs et al. (1933)]. They were developed by the National Advisory Committee for Aeronautics (NACA), the predecessor of NASA. The shape of the NACA airfoils is described by a numerical code, comprised of a series of digits. The parameters can be entered into analytical equations to precisely generate the cross-section of the airfoil and calculate its properties. Today, airfoil design has, in many ways, returned to an earlier time before the NACA families were created as the computational resources available now allow the designer to quickly design and optimise an airfoil specifically tailored to a particular application rather than making a selection from an existing family.

Parameterisation methods for representing curves and surfaces have become prevalent in the CAD community. The Bézier curve is one of the simplest forms among the parameterisation methods. It is an effective way for representing simple curves in a compact fashion. They were named after the French engineer Bézier (1968), who used

---

<sup>1</sup><http://www.3ds.com/>

<sup>2</sup>[http://www.plm.automation.siemens.com/en\\_us/products/nx/index.shtml](http://www.plm.automation.siemens.com/en_us/products/nx/index.shtml)

<sup>3</sup><http://www.ptc.com/>

<sup>4</sup><http://www.opencascade.org/>

them to design automobile bodies and made them widely publicised. However, The Bézier form is inefficient for high degree complex curves. A B-spline is a generalisation of a Bézier curve which could represent high degree curves without using inefficient high degree polynomials. B-splines can be evaluated in a numerically stable way by the de Boor algorithm [de Boor (1971)]. Computing the derivatives at a point on a Bézier curve or B-spline is easy. More detailed mathematical descriptions of B-spline and its derivatives are given in Section 3.3. B-spline curves are polynomial curves. While they are flexible and have many nice properties for curve design, they are not able to represent conic curves, including the circle. The nonuniform rational B-spline (NURBS) is a generalisation of the B-spline to address the problem. It can represent most parametric and implicit curves and surfaces accurately [Piegl and Tiller (1997)]. Bézier curve, B-splines and NURBS are all parameterisation methods for geometric construction and manipulation. They provide the flexibility and intuitiveness to design a large variety of shapes and reduce the memory consumption when storing shapes.

There are also parameterisation techniques for complex aerospace models, none of them in domination though. Among them are the basis vector approach [Pickett et al. (1973)], discrete element approach [Belegundu and Rajan (1988)], the domain element approach [Leiva and Watson (1999)], the partial differential equation approach [Bloor and Wilson (1989)] and the free form deformation [Sederberg and Parry (1986)]. The choice of shape parameterisation techniques are of fundamental importance in the design and optimisation process. An extensive survey of techniques and their suitability for shape optimisation can be found in [Samareh (2001)].

### **2.1.3 CFD and downstream applications**

The Computational Fluid Dynamics (CFD) is another element that usually functions as an integrated part of systematic design optimisation in which it is used to obtain flow predictions. The fluid behaviour can be described by the Navier-Stokes equations, which account for the fluid compressibility and viscosity allowing the modelling of shock, boundary layer separation, etc. In aerospace applications, these effects have significant impacts on the designs. The Navier-Stokes equations are spatially and temporally dependent partial differential equations whose solution are still beyond current computational capabilities in spite of recent increases in computing power. As a result, two important simplifications are common in practice. Reynolds Averaged Navier-Stokes (RANS) equations use turbulence models to estimate fluctuations. It still allows compressibility and viscosity to be considered, resulting in accurate estimation in most cases. The solution of RANS can still take considerable amount of computational power, for example, [Duchaine et al. (2009)]. If inviscid flow is assumed, Euler equations will result. The solution of an Euler system can be faster. However by definition inviscid flow model neglects the dissipative, transport phenomena of viscosity, mass diffusion, and thermal



conductivity, thus it could lead to inaccurate prediction when these phenomena are significant.

## 2.2 Design optimisation

Design optimisation is the practice of improving existing designs with certain goals and within constraints, such as robustness, safety and cost. In the simplest case, this means solving problems in which one seeks to minimise or maximise a real function by systematically choosing the values of real or integer variables from within an allowed set. Designers have a very wide range of optimisation methods available, each suitable for a subset of optimisation problems. Therefore it is important to be aware of the category to which the optimisation problem at hand belongs. The optimisation problems can be categorised in regard of the input(s), output(s) or functional relationship between them.

In most aerospace design cases, continuous numerical input optimisation problems are often dealt with. Discrete numerical or nonnumeric inputs are not uncommon: for example, if the objective is to optimise number of blades on an engine compressor disk, or to choose from a ventral or side-mounting engine intake installation.

According to the number of outputs, the optimisation problem can be categorised into single objective or multi-objective optimisation problems. A multi-objective optimisation maximises or minimises more than one objective function at the same time. Having more than one objective to an optimisation problem adds complexity because different objectives often conflict. The solution of multi-objective problems often leads to a trade-off between designs. An important concept related to this trade-off is the Pareto set. Designs in the Pareto set are those wherein any change in design inputs to improve any single objective would make others worse. The designs in the Pareto set are called Pareto optimal.

Optimisation problems can also be classified depending on the functional relationship between the inputs and outputs. They can be linear or nonlinear, deterministic or stochastic. Linear problems are those in which the objective function is linear and subject to linear equality and inequality constraints, whereas nonlinear optimisation refers to the general case in which the objective function or the constraints or both are nonlinear in the design variables. In aerospace design optimisation, many outputs come from deterministic nonlinear computer codes. Stochastic problems are those in which some of the constraints or parameters depend on random variables, for example, when taking manufacturing variability or changing working environment load into consideration, the problems often become nonlinear stochastic.

For each type of optimisation problem, the optimisation community has been trying to develop and improve optimisation methods. Usually, the optimisation method is an

iterative one for finding the minimum, i.e., given an initial point, an iteration sequence will be generated by the optimisation method, the last point being the optimal solution of the problem. A practical method should move steadily towards the neighbourhood of the minimum, and converge rapidly to it. When a convergence rule is satisfied, the iteration will be terminated.

### 2.2.1 Linear and nonlinear programming

For linear problems, various linear programming methods have been proposed. The simplex method was one of the most efficient basis-exchange pivot algorithm for a great majority of practical problems [Bixby (1991)]. First put forward by Dantzig and Orchard-Hays (1954), it was listed as one of the “top 10 algorithms of the twentieth century” by the journal *Computing in Science and Engineering* [Dongarra and Sullivan (2000)], with quick-sort algorithm and fast Fourier transform. The simplex method relies on noticing that the objective function’s maximum must occur on a corner of the space bounded by the constraints of the feasible region. Criss-cross algorithm is another family of basis-exchange pivoting algorithm, first published independently by Terlaky (1985) and Wang (1987)<sup>5</sup>. Fukuda and Terlaky (1997) present mathematical ideas and proof techniques behind the finite criss-cross pivot methods. Interior-point method is another family of linear programming algorithms, prominently among which are ellipsoid method of Khachian (1980) and projective algorithm of Karmarkar (1984). Vanderbei (2008) provides a thorough treatment of linear programming, quadratic programming, and convex optimisation<sup>6</sup>.

There is a vast range of methods for nonlinear problems; they are collectively known as nonlinear programming. They can be classified according to whether they require derivative information of the function. Those methods that do not require gradient information are sometimes referred to as population-based methods or zeroth order methods. Methods that require gradient information are known as gradient-based algorithms.

### 2.2.2 Population-based algorithms

The most basic form among population-based algorithms is probably the line search, which optimises univariate functions. In the line search, an initial search interval which contains the minimiser need to be determined; then sectioning techniques are employed to reduce the interval iteratively until the length of the interval is less than some given tolerance. The golden section method and the Fibonacci method are both section methods. The golden section method derives its name from the fact that the algorithm

---

<sup>5</sup>Full texts of either article are not publicly available.

<sup>6</sup>And this book is available online

maintains the function values for triples of points whose distances form a golden ratio. The Fibonacci search technique narrows down possible locations with the aid of Fibonacci numbers.

Evolutionary algorithms are zeroth order methods as well. Among the large family of evolutionary methods are evolutionary strategies, genetic algorithms and simulated annealing [Bäck et al. (1997)]. Such classes of methods only need to evaluate the functions and has wide applications, for example, [Song and Keane (2005)]. Especially it is suitable to non-smooth problems, problems with complicated derivative expressions, and problems whose derivatives cannot be easily determined.

Evolution strategies (ES), developed by Rechenberg (1973), and extended by Ostermeier et al. (1994) and Rudolph (1991) were initially designed with the goal of solving difficult discrete and continuous parameter optimisation problems. Evolution strategies use natural problem-dependent representations, and in common with evolutionary algorithms, the operators are applied in a loop. An iteration of the loop is called a generation. The sequence of generations is continued until a termination criterion is met. Beyer and Schwefel (2002) provided a comprehensive introduction into ES history, algorithms and design principles.

Genetic algorithms, became popular through the work of Holland (1975), are classified as global search heuristics. The techniques are inspired by the evolutionary biology such as inheritance, mutation, selection, and crossover. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated; multiple individuals are selected from the current population based on their fitness, and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. [Goldberg (1989)], a classic in the field of genetic algorithms, introduces the theory, operation, and applications<sup>7</sup>. [Konak et al. (2006)] presented recent advances in using genetic algorithms for multiple-objective optimisation.

Simulated annealing [Kirkpatrick (1984)] is another generic heuristics for the global optimisation problem. The method is an analogy with the physical annealing process in metallurgy. Each point of the search space is analogous to a state of some physical system, and the objective function to be minimised is analogous to the internal energy of the system in that state. The goal is to bring the system to a state with the minimum possible energy. In each step of the SA algorithm, the current solution is replaced by a random neighbour, chosen with a probability that depends on the difference between

---

<sup>7</sup>The author Goldberg was a student of Holland

the corresponding function values and on a global parameter  $T$ .  $T$  physically represents the temperature which gradually decreases during the annealing process. The evolution of the state  $s$  depends crucially on the value of  $T$ . The evolution is coarser when the  $T$  is larger and finer when the  $T$  is smaller. Permitting uphill steps saves the method from becoming stuck at a local minima.

### **2.2.3 Gradient-based algorithms**

Methods that require the derivative information of the objective function are gradient-based algorithms. The steepest descent method [Fletcher and Powell (1963)], which uses the negative gradient as its descent direction is probably the most fundamental and simplest in this category. Newton's method requires second order derivatives information to approximate the objective function in the neighbourhood of the current iterate, and then minimise the approximation. The second order gradient information, usually represented by the Hessian matrix, can be made available using finite differencing or via an adjoint code. However, the Hessian can be computationally expensive to obtain and sometimes indefinite, which can make the Newton method inapplicable. Therefore, many modifications to Newton's method exist. The conjugate gradient method [Hestenes and Stiefel (1952)] only requires the first-order derivatives but overcomes the steepest descent method's shortcoming of slow convergence. At the same time, the method need not save and compute the second-order derivatives. Therefore, it is widely used to solve large scale optimisation problems. Trust-Region methods approximate only a certain region (the so-called trust region) of the objective function with a model function (often a quadratic) as opposed to the entire function as with the Newton's method [Conn et al. (2000)]. When an adequate model of the objective function is found within the trust region then the region is expanded. Conversely, if the approximation is poor, then the region is contracted. The method can be applied to find the global optimum. There is also a large family of methods for constrained optimisation problems, for example, penalty function methods, feasible sequential quadratic programming [Boggs and Tolle (2008), Lawrence (1998)] and feasible direction methods. Many hybrid methods that combine different methods have been developed to achieve better results, for example, [Jian (2005)].

### **2.2.4 Computational engineering design and optimisation framework**

**Computer based optimisation** In an ideal automated design optimisation framework, usually a parametric geometry engine is used to generate parametric models. The models are to be meshed and transferred into CFD/FEA or other analysis modules. The analysis results are sent back to the optimiser. If the preset optimisation objective is reached, the process ends. If not, the optimisation algorithm will propose further design

candidates. The design–analysis–optimisation cycle is repeated until the goal is reached or time/computing budget is used up.

The increasing complexity of engineering systems has sparked rising interest in multi-disciplinary optimisation (MDO). MDO is a field of optimisation that solves complex design problems in which there is strong interaction between disciplines [Price et al. (2010)]. Only by considering the fully coupled system can an optimal design emerge. The primary challenges in MDO are computational expense and organisational complexity. Sobieszcanski-Sobieski and Haftka (1997) surveyed various methods used by different researchers to address these challenges. [Vanderplaats (2007)] systematically described MDO techniques, with emphasis on application to engineering design. Yi et al. (2008) compared seven common MDO methods with mathematical examples. [Simpson and Martins (2011)] is a recent examination of MDO’s current and future role in designing complex engineered systems.

Surrogate modelling and sampling plan design are not optimisation techniques in themselves. However they are closely related to optimisation because they reduce the computational cost of optimisation by transferring computationally expensive problems into simpler tasks which could be solved by the algorithms outlined in the previous sections [Keane and Scanlan (2007)]. Section 2.3 and Section 2.4 provide more detailed reviews on these two areas.

## 2.3 Surrogate modelling

Although computing power has improved significantly in the last few decades, the computational cost associated with engineering design optimisation remains a challenge. This is mainly due to the engineers’ desire to model engineering systems with higher fidelity. For example, the Reynolds Averaged Navier-Stokes (RANS) equations that are used to describe high-fidelity simulation models can take hours to solve, depending on the size and nature of the problem [Duchaine et al. (2009)]. This is compounded by the fact that it is common that an optimisation algorithm can make many function evaluations before an optimal solution is found. Optimisation can lead to prohibitive computational requirements when high-fidelity simulation is used. [Schmit Jr. and Farshi (1974)] provided an early example of the idea of surrogate modelling. The basic idea of surrogate modelling is to construct an approximation of the expensive objective function from the data generated by a sampling plan. The optimisation is then applied to the approximation model, therefore reducing the computational cost. In many cases, data obtained during the optimisation process are used to update the surrogate model in order to make it more accurate. The process continues until certain convergence criteria

are met. The technique, mainly used as an approximation technique to simulate computationally expensive codes, has gained wide notice by both academia and industry, and has been an cutting-edge research area in the last decade.

In this section, response surface methodology, radial basis function networks, Kriging and support vector regression are reviewed in turn.

### 2.3.1 Response surface methodology

The Response Surface Methodology (RSM) was first proposed by Box and Wilson (1951). It is derived from the least squares method. For a first order model

$$\hat{f}(\mathbf{x}) = w_0 + [w_1, \dots, w_k]^T \mathbf{x}, \quad (2.1)$$

the coefficients  $\mathbf{w} = [w_0, w_1, \dots, w_k]$  can be estimated by

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \quad (2.2)$$

where  $\mathbf{X}$  is the sampling plan and  $\mathbf{y}$  is the output vector.

Higher order polynomial models are used to approximate nonlinear functions. RSM has been used in a variety of applications including multidisciplinary optimisation [Korngold and Gabriele (1997)] and decision-based designs [Lewis and Mistree (1998)], in which rational reaction sets are approximated by the RSM. Detailed application of the methods and a MATLAB implementation can be found in [Forrester et al. (2008)].

### 2.3.2 Radial basis function networks

Radial Basis Function (RBF) networks can produce good approximations to arbitrary functions. Mathematically, the RBF network model has the general form

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^n w_i \psi(\|\mathbf{x} - \mathbf{c}_i\|) \quad (2.3)$$

where  $\|\cdot\|$  represents the Euclidean norm,  $\mathbf{c}_i$  denotes the  $i^{th}$  basis function centre and  $\psi$  represents the basis function, which can take various forms. Common choices of basis functions include

- linear  $\psi(r) = r$
- cubic  $\psi(r) = r^3$
- thin plate spline  $\psi(r) = r^2 \ln r$

- Gaussian  $\psi(r) = e^{-\frac{r^2}{2\sigma^2}}$
- multi-quadric  $\psi(r) = (r^2 + \sigma^2)^{\frac{1}{2}}$

where  $r$  is equal to  $\|\mathbf{x} - \mathbf{c}_i\|$ . Usually  $\mathbf{c}_i$  is chosen to be the same as  $\mathbf{x}_i$  for calculation convenience. In order to estimate the basis function weights  $\mathbf{w}$ , the interpolation condition is used.

$$\hat{f}(\mathbf{x}_j) = \sum_{i=1}^n w_i \psi(\|\mathbf{x}_j - \mathbf{x}_i\|) = y_i \quad j = 1, 2, \dots, n. \quad (2.4)$$

Defining the Gram matrix  $\Psi_{i,j} = \psi(\|\mathbf{x}_j - \mathbf{x}_i\|)$ , Equation 2.4 can be written as  $\Psi \mathbf{w} = \mathbf{y}^T$ . Provided the inverse of  $\Psi$  exists, the coefficients can be determined by  $\mathbf{w} = \Psi^{-1} \mathbf{y}^T$ .

The prediction at an arbitrary point  $\mathbf{x}$  is

$$\hat{f}(\mathbf{x}) = \phi \mathbf{w} = \phi \Psi^{-1} \mathbf{y}^T \quad (2.5)$$

where

$$\phi = [\psi(\|\mathbf{x} - \mathbf{x}_1\|), \psi(\|\mathbf{x} - \mathbf{x}_2\|), \dots, \psi(\|\mathbf{x} - \mathbf{x}_n\|)].$$

For Gaussian and multi-quadric basis functions, it is also desired to find an optimised value of the parameter  $\sigma$  that can minimise the (estimated) generalisation error of the model. This can be done via a cross-validation process. In the cross-validation process, the training data are randomly split into equal subsets, then each of the subsets are removed in turn and the surrogate model is fitted to the remaining data. The error  $E$  between the predictor and the points in the subset that is set aside at each iteration can be used as a measure of the quality of the surrogates. A list of values of  $\sigma$  can be populated beforehand and each used as the model parameter in turn. The value of  $\sigma$  that delivers the smallest error  $E$  can be chosen as the optimised value.

A MATLAB implementation of the radial basis function and the cross validation process can be found in [Forrester et al. (2008)]. The basic RBF can be enhanced to include regression by introducing a regularisation parameter  $\lambda$  [Poggio and Girosi (1990)]. The gradient information can also be utilised to build the RBF model [Kampolis et al. (2004)]. RBFs have been extensively applied to engineering problems, such as the parametric geometry repair model in [Sóbester and Keane (2006)] and the construction of meta-modelling of combined discrete/continuous responses in [Meckesheimer et al. (2001)].

### 2.3.3 Kriging

The kriging method was pioneered by geological statistician Krige to interpolate the value of a numerical field (e.g., the mineral concentrations or the elevation of the landscape, as a function of the geographic location) at an unobserved location from observations of its value at nearby locations [Cressie (1990)]. It was then introduced and popularised as an alternative method of creating surrogate models of deterministic computational experiments [Matheron (1963), Sacks et al. (1989)]. It has been extensively used in various applications, for example, aerodynamics [Hoyle et al. (2006)] and multi-objective optimisation [Keane (2006)].

Kriging views the response  $y_i$  from a statistical perspective, as if it were a realisation of a stochastic process. The correlations between random variables  $y_i$  and  $y_j$  are given by

$$\text{corr}(y_i, y_j) = \exp \left( - \sum_{l=1}^k \theta_l \|x_i^{(l)} - x_j^{(l)}\|^{p_l} \right), \quad (2.6)$$

where the hyperparameters  $\theta_l$  and  $p_l$  in Equation 2.6 are then calculated using maximum likelihood estimation [Jones (2001)]. A recent discussion of kriging hyperparameter tuning strategies can be found in [Toal et al. (2008)], in which how often and how well it is necessary to tune the hyperparameters of a kriging model as it is updated during an optimisation process is investigated.

### 2.3.4 Support vector machine and support vector regression

Support Vector Machine (SVM) is a supervised learning method firstly proposed by Cortes and Vapnik (1995) at the AT&T Bell Laboratories, which was developing optical character recognition (OCR) at that time. SVR was a classification technique based on the idea of the empirical risk minimisation principle. Classifying data point is a common task in machine learning: suppose some given data points each belong to one of two classes, the goal of classification is to decide which class a new data point will be in. The SVM solves this problem by seeking a “maximum-margin hyperplane” which separates the two classes of data using the given data points. Maximum-margin hyperplane is defined as one that has the largest distance to the nearest training data point of any class. Since in general the larger the margin the lower the generalisation error of the classifier, maximum-margin hyperplane provides optimal classification. For a  $k$ -dimensional space, a  $k - 1$  dimensional maximum-margin hyperplane is needed. Figure 2.1 illustrates the idea of class separation and maximum-margin hyperplane in a 2-dimensional space. In the figure,  $H_3$  doesn't separate the two classes.  $H_1$  does, with a small margin and  $H_2$  with the maximum margin.

The samples of the given data points that define the hyperplane are called support vectors. A comprehensive understanding of SVM requires the synthesis of a wide range



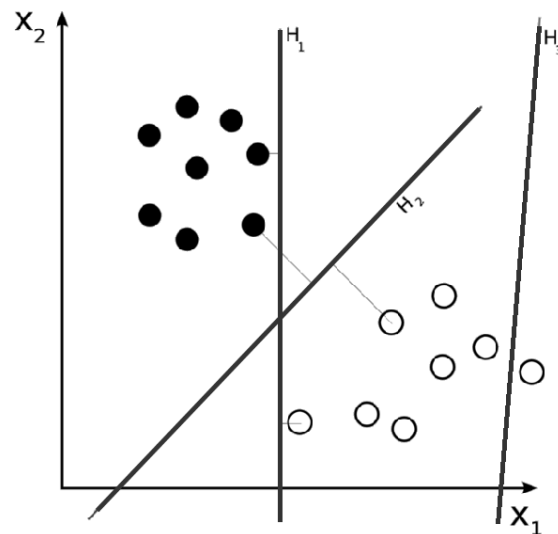


FIGURE 2.1: Illustration of hyperplanes in a 2-dimensional space

of topics, including dual representations, feature spaces, learning theory, optimisation theory, and algorithmics [Vapnik (1995), Cristianini and Shawe-Taylor (2000), Sánchez et al. (2011)].

To allow for mislabelled data, Cortes and Vapnik (1995) suggested soft margin method as a modified maximum-margin method. If there is no hyperplane that can split the given data, the soft margin method chooses a hyperplane that splits the examples as cleanly as possible, while still maximising the distance to the nearest cleanly split examples.

While SVM are becoming competitive for OCR [Joachims (1998), Sebastiani (2002)] and other object recognition tasks [Furey et al. (2000), Pan et al. (2002)], excellent performance has been achieved in regression applications as well. Promising empirical performance is shown in [Vapnik et al. (1996), Gunn (1998), Schoelkopf et al. (1997)]. The support vector regression (SVR) algorithm has been developed by Drucker et al. (1997). In the regression case the loss function used only penalises errors greater than a threshold. Such a loss function typically leads to a sparse representation of the decision rule giving significant algorithmic and representational advantages. The application of the support vector approach to regression adds the capability to approximate functions. Initial comparison of SVR with other methods has been made using 26 engineering analysis functions by Clarke et al. (2005), and SVR has achieved more accurate and more robust function approximations. Many works contribute to make large-scale SVR learning practical, for example, [Quan et al. (2004)]. The two main references used by this work for this approach are [Forrester et al. (2008)] and [Smola and Schölkopf (2004)].

To allow for nonlinearity in classification and regression, kernel trick originally proposed by Aizerman et al. (1964) is introduced into SVM/SVR. The kernel trick basically replace every dot product by a nonlinear kernel function. This allows the algorithm to fit the

maximum-margin hyperplane in a transformed feature space [Boser et al. (1992)]. More details of kernel trick is presented in Section 3.2.4.

## 2.4 Sampling plans

Discrete observations or samples are basic building blocks for surrogate modelling. A well-designed sampling plan is a precondition for any surrogate model. A well-posed surrogate model may not generalise well if the sampling plan fails to explore certain regions of the input space. The accuracy of the prediction is restricted by the density of the sampling plan as well. However the number of sampling points is inevitably limited by the computing budget. Thus, it is important to make wise decision on the sampling plan.

Perhaps the full factorial sampling technique is the most straightforward way of sampling a design space. Such a sampling plan forms a rectangular grid of points in a uniform fashion. It is easy to implement but when projected to the axes, points will overlap. Intuitively, this is a waste of the computing budget assigned for design space exploration. A good sampling plan should cover the design space in a thorough and uniform fashion. The requirement of a uniform projection onto each dimension of the design space leads to the idea of Latin hypercube sampling, in which the sample points align in such a way that there is only one sample in each row and each column [McKay et al. (1979)]. It is also desired that the sampling points spread in the design space as uniformly as possible. The maximin metric, originally introduced by Johnson et al. (1990) and further elaborated by Morris and Mitchell (1995) and Ye et al. (2000), is widely used to evaluate the uniformity of a sampling plan. [Forrester et al. (2008)] illustrates how optimised space-filling Latin Hypercube can be obtained by evolutionary operation using maximin criteria, and the method is used in this work.

## 2.5 Knowledge-based systems

Knowledge-based systems are expert systems based on the methods and techniques of artificial intelligence. La Rocca and van Tooren (2010) defined knowledge-based engineering system as systems that identify, captures and reuses design, product and process knowledge in an integrated way. The application of knowledge-based systems reduce time and cost for engineering applications, automating repetitive design tasks, and support conceptual design activities. Eom (1992) surveyed expert system applications in production and operations management. Maher et al. (1984) described tools and techniques for knowledge-based expert systems for engineering design. Common knowledge representation techniques include frames, rules, tagging, and semantic networks. Oxman and Oxman (1991) provided a theoretical perspective of knowledge-based CAD.

Different from the common knowledge-based engineering approach above-mentioned, a branch of research deals with incorporating prior knowledge into surrogate models. If prior knowledge of the black-box function exists, this can be used to enhance the accuracy of the surrogate models. Wang and Zhang (1997) developed a knowledge-based neural network model for microwave design. Problem specific knowledge in the form of generic empirical functions was included to improve the accuracy of the surrogate. Leary et al. (2003) extended this approach by incorporating low fidelity information inside the network. They also used a new knowledge-based kriging method instead of an artificial neural network. Lauer and Bloch (2008) proposed methods of incorporating prior knowledge in linear programming support vector regression. The type of knowledge included particular points with known values and derivatives of the function. Bloch et al. (2008) also proposed an algorithm for adding potential support vectors into the surrogate. These are, in essence, methods of adding constraints to the linear programming SVR optimisation problem.

## 2.6 Geometry repair

As briefly mentioned in Section 1.1, some literature, for example, [Butlin and Stops (1996)], use the word “repair” to refer to the operation onto a geometry to get it into a form suitable for engineering analysis. Similarly, Hu et al. (2002) use the word “geometry clean-up” to refer to the removal of geometry inconsistencies including common edge defined twice, small gaps, holes, and overlapping of faces. Another similar concept is “de-featuring” [Lee et al. (2003), Gopalakrishnan and Suresh (2007)], which means suppressing small or irrelevant features for speeding-up downstream processes. None of these has directly addressed the control of the tradeoff between robustness and flexibility of the parametric geometry generation, and should not be confused with the term ‘geometry repair’ that is used in the thesis which refers to the automatic repair of infeasible design from engineering design point of view.

Recent research suggested that a supervised learning system can be attached to the design system to capture some of the engineering and geometrical judgment of the designer. Thereafter, the knowledge could be used to repair design variable sets that lead to infeasible geometry models. In [Sóbester and Keane (2006)], the idea was tested by building a radial basis function (RBF) supervised learning system into the design process of a jet engine nacelle. The approach is robust and effective for models described by a moderate number of design variables (ten design variables are used in [Sóbester and Keane (2006)] for defining a jet engine nacelle). However, the amount of training data required to build up an accurate surrogate increases exponentially with the number of design variables, which makes the amount of work required to prepare the training data become less affordable for larger scale conceptual design processes. For example, a typical aircraft design may be summarised at the concept design stage by nearly 100

variables [Keane and Nair (2005b)] and a wing characterised by 30 [Walsh et al. (2000)]. Therefore, in order to make the approach applicable to a broader range of real life problems, it is necessary to improve the above mentioned approach so that it becomes a more affordable one.



## Chapter 3

# Methodology

In this chapter, the essential elements of the proposed knowledge-based geometry repair system are reviewed in detail. Firstly, the way that physical and engineering base knowledge can be gathered and transferred to useful information are discussed first in Section 3.1. Section 3.2 explains in further detail how to use the SVR to represent the knowledge at hand. Section 3.3 introduces B-spline representation of curves which is used in subsequent chapters. In Section 3.4, the concept of geometry repair and the process of search for a repair alternative are explained in detail.

### 3.1 Knowledge representation

An engineer's judgement and expert knowledge of the feasibility of a model can be drawn from various sources such as

- explicit rules, discovered by using engineering or geometrical judgment; these rules include equality and inequality constraints, parameters and engineering laws, etc.
- assessment of individual design cases by an expert engineer, and
- computational analysis results (from CFD, FEA, etc.).

There are no universally applicable schemes for incorporating engineering knowledge into a design system. This work shows the possibility of transferring some knowledge into explicit functions and incorporate the knowledge into a regression model. Furthermore, knowledge of specific designs taken directly from individual engineers, existing designs or computational analysis results can be mapped to design variable sets [Sóbester and Keane (2006)]. The explicit rules can be transformed into penalty functions. The penalty function method is commonly used in design optimisation to transform engineering constraints into objective functions, onto which various optimisation techniques can then

be applied. The simplest penalty approach in the optimisation literature is to attach a large penalty constant to the original objective function whenever any constraint is violated. Although simple to apply, the approach causes a discontinuity in the shape of the penalised objective at the constraint boundary and takes no account of the number of constraint violations. These limitations can be mitigated by some modifications. First, the penalty can be multiplied by the degree of violation of the constraint for continuity. Secondly, a separate penalty function may be applied to each violated constraint. In this work, both modifications are adopted and the penalty is used to indicate the degree of violation of engineering constraints, in other words, the feasibility of the design. The more constraints are violated or the worse a single constraint is violated, the higher the penalty is and the less feasible the design becomes. In this work, this idea is applied, by constructing four penalty functions [Keane and Nair (2005c)] resulting from practical considerations and physics-based analysis. The penalty functions take a set of design variables as input and generate a numerical penalty value for this set as output. The resulting data are collected, stored and used to train a surrogate. As a result, the surrogate ‘learns’ from the penalty functions, which are representations of engineering knowledge.

When knowledge is drawn from the physical properties of the geometric model in question (for example, the aerodynamics or the mechanics), these can be calculated from engineering laws. More commonly in optimisation design practice, they are predicted by numerical analysis codes. These codes can be viewed as black box functions, whose input (parametric geometries defined by design variables) and output (the physical properties in question) are known. Once the data have been collected, they can be utilised in the same way the data from penalty functions were utilised to train the surrogate.

Furthermore, knowledge of specific designs can be captured directly from case-by-case assessment by engineers. Experienced engineers can exercise their own judgement and score a design based on its feasibility, for example, from 0 to 10. The score and its corresponding design variable set can be stored and used as training data for the surrogate. This process was demonstrated by [Sóbester and Keane (2006)].

Figure 3.1 summarises the knowledge representation process.

## 3.2 Using support vector regression as the knowledge surrogate

Having reviewed surrogate modelling literature in Section 2.3, it is noticed that support vector regression can be used to avoid difficulties of using linear functions in the high dimensional feature space, because the model produced by SVR only depends on a subset of the training data. The SVR model ignores any training data that are close (within

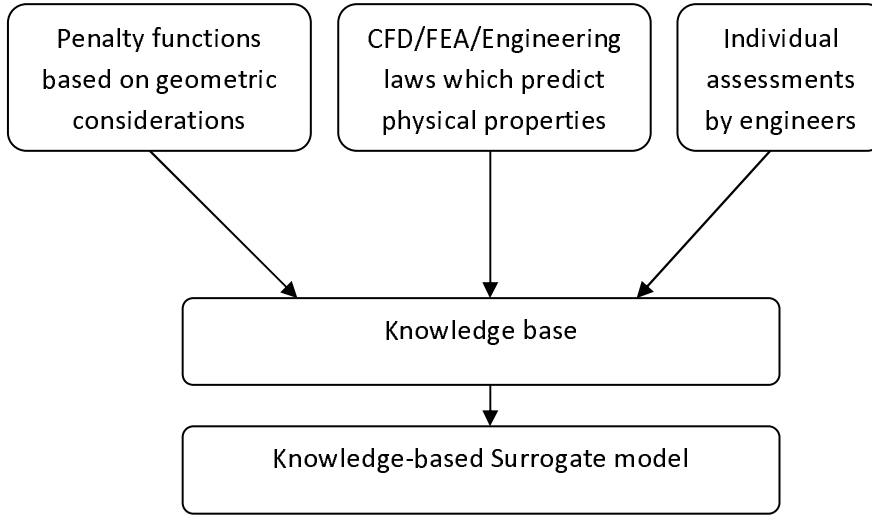


FIGURE 3.1: Knowledge representation

a threshold  $\epsilon$ ) to the model prediction and does not depend on the dimensionality of the input space. Because of these, it is expected that SVR will have advantages in the high dimensionality space, and the sparse representation of the SVR model can give algorithmic and representational advantages. This can be a useful feature in real life design applications where the number of design variables is high. In this section, the basic ideas, mathematics and implementation of the SVR method are examined. The SVR method is used as the main tool for knowledge base construction later discussed in Section 3.4 and is tested in later chapters.

### 3.2.1 The basic idea

Suppose the training data  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ , where  $\mathbf{x}_i \in \mathbb{R}^k, \forall i \in \{1, \dots, n\}$ , has been obtained and the outputs from the analysis is vector  $\mathbf{y} = \{y_1, y_2, \dots, y_n\}$ . Support vector regression allows specification of a margin  $\epsilon$  within which prediction errors could be tolerated. Thus, the prediction function  $f(x)$  will have deviation no larger than  $\epsilon$  from the actually obtained function values for all the training data. On the other hand, it is desired to minimise the model complexity. For example, consider the first order linear regression model

$$f(x) = \mathbf{w}^T \mathbf{x} + \mu \quad (3.1)$$

where  $\mathbf{w}$  is the coefficient vector  $\{w_0, w_1, \dots, w_k\}$  and  $\mu$  is the bias. Its model complexity can be measured by the norm of  $\mathbf{w}$ , i.e.  $\|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w}$ .



The problem described above can be cast in the form of a convex optimisation problem:

$$\begin{aligned} & \text{minimise} && \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{subject to} && \begin{cases} y_i - \mathbf{w}^T \mathbf{x}_i - \mu \leq \epsilon \\ \mathbf{w}^T \mathbf{x}_i + \mu - y_i \leq \epsilon. \end{cases} \end{aligned} \quad (3.2)$$

Notice that the above optimisation problem may not be feasible when such a function  $f$  which approximates all pairs  $(\mathbf{x}_i, y_i)$  within  $\epsilon$  precision does not exist. To tackle this, slack variable pairs  $\xi^+, \xi^-$  are introduced to relax the constraints in the original problem in case Equation 3.2 becomes infeasible. Of course it is desired that these slack variables to be as small as possible. Thus, Equation 3.2 can be transformed into:

$$\begin{aligned} & \text{minimise} && \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^k (\xi_i^+ + \xi_i^-) \\ & \text{subject to} && \begin{cases} y_i - \mathbf{w}^T \mathbf{x}_i - \mu \leq \epsilon + \xi_i^+ \\ \mathbf{w}^T \mathbf{x}_i + \mu - y_i \leq \epsilon + \xi_i^- \\ \xi_i^+, \xi_i^- \geq 0 \end{cases} \end{aligned} \quad (3.3)$$

The constant  $C \geq 0$  in Equation 3.3 is a user-defined parameter, which determines the trade-off between model complexity and the amount up to which excessive error can be tolerated. When  $C$  approaches 0, the latter term in the optimisation objective,  $C \sum_{i=1}^k (\xi_i^+ + \xi_i^-)$ , also approaches 0 since the summation of slack variable pairs is always finite. Thus the optimisation focuses on minimising  $\frac{1}{2} \|\mathbf{w}\|^2$ , and results in a flat prediction. On the other hand, a larger constant  $C$  will lead to a closer fitting of the training data, since more emphasis is put on minimising  $\sum_{i=1}^k (\xi_i^+ + \xi_i^-)$ . So  $C$  can be regarded as a cost index: the higher  $C$  is, the more costly prediction error becomes.

### 3.2.2 Finding $\mathbf{w}$

Having understood the basic idea of support vector regression, a practical solution to the above constrained optimisation problem is pursued. The key idea is to introduce two sets of Lagrange multipliers:

$$\begin{cases} \eta_i^+ \geq 0 \\ \eta_i^- \geq 0 \end{cases} \quad \text{and} \quad \begin{cases} \alpha_i^+ \geq 0 \\ \alpha_i^- \geq 0 \end{cases}$$

which correspond to the constraints

$$\begin{cases} y_i - \mathbf{w}^T \mathbf{x}_i - \mu \leq \epsilon + \xi_i^+ \\ \mathbf{w}^T \mathbf{x}_i + \mu - y_i \leq \epsilon + \xi_i^- \end{cases}$$

and

$$\begin{cases} \xi_i^+ \geq 0 \\ \xi_i^- \geq 0 \end{cases}$$

in Equation 3.3 respectively. The Lagrange function is then given by the original objective function minus the sum of all products between Lagrange multipliers and corresponding constraints:

$$\begin{aligned}
L = & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^k (\xi_i^+ + \xi_i^-) \\
& - \sum_{i=1}^k (\eta_i^+ \xi_i^+ + \eta_i^- \xi_i^-) \\
& - \sum_{i=1}^k \alpha_i^+ (\epsilon + \xi_i^+ - y_i + \mathbf{w}^T \mathbf{x}_i + \mu) \\
& - \sum_{i=1}^k \alpha_i^- (\epsilon + \xi_i^- + y_i - \mathbf{w}^T \mathbf{x}_i - \mu).
\end{aligned} \tag{3.4}$$

Duality theory in convex optimisation states that  $L$  should be minimised with respect to the primal variables  $\mathbf{w}, \mu$  and  $\xi^\pm$  while at the same time maximised with respect to the dual variables, which are the Lagrange multipliers  $\eta^\pm$  and  $\alpha^\pm$  in this case.

To find  $\mathbf{w}$ , the fact that the solution must be at a saddle point is exploited. Applying the saddle point condition, all partial derivatives with respect to the primal variables equal zero. In particular,

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^k (\alpha_i^+ - \alpha_i^-) \mathbf{x}_i = 0. \tag{3.5}$$

Rearranging (3.5)

$$\mathbf{w} = \sum_{i=1}^k (\alpha_i^+ - \alpha_i^-) \mathbf{x}_i. \tag{3.6}$$

Substituting (3.6) into (3.1)

$$\begin{aligned}
\hat{f}(x) &= \sum_{i=1}^k [(\alpha_i^+ - \alpha_i^-) \mathbf{x}_i] \cdot \mathbf{x} + \mu \\
&= \sum_{i=1}^k (\alpha_i^+ - \alpha_i^-) (\mathbf{x}_i \cdot \mathbf{x}) + \mu.
\end{aligned} \tag{3.7}$$

The above process is called the support vector expansion, in which  $\mathbf{w}$  is completely described as a linear combination of the training data. Thus, it is not necessary to compute  $\mathbf{w}$  explicitly to get the prediction model. Notice that  $\alpha^\pm$  remain unknown for the time being. They are computed by the quadratic programming approach, which is briefly described later in Section 3.2.7.

### 3.2.3 Finding $\mu$

After  $\mathbf{w}$  has been found by the support vector expansion, it is necessary to find the bias term  $\mu$  to complete (Equation 3.1). The Karush–Kuhn–Tucker conditions [Karush (1939)] state that the product between dual variables and the constraints vanishes at the solution point:

$$\begin{cases} \alpha_i^+(\epsilon + \xi_i^+ - y_i + \mathbf{w}^T \mathbf{x}_i + \mu) = 0 \\ \alpha_i^-(\epsilon + \xi_i^- + y_i - \mathbf{w}^T \mathbf{x}_i - \mu) = 0 \end{cases} \quad (3.8)$$

and

$$\begin{cases} \xi_i^+(C - \alpha_i^+) = 0 \\ \xi_i^-(C - \alpha_i^-) = 0. \end{cases} \quad (3.9)$$

Notice that if

$$\epsilon + \xi_i^+ - y_i + \mathbf{w}^T \mathbf{x}_i + \mu = 0$$

or rearranging it as

$$y_i - (\mathbf{w}^T \mathbf{x}_i + \mu) = \epsilon + \xi_i^+$$

the prediction value lies at the upper boundary of the allowable region; on the other hand, if

$$\epsilon + \xi_i^- + y_i - \mathbf{w}^T \mathbf{x}_i - \mu = 0,$$

the prediction value lies at the lower boundary. These two opposite scenarios cannot happen simultaneously at the same point, so the above two terms cannot be both simultaneously zero. Now examining Equation 3.8), to make sure it holds, either  $\alpha_i^+$  or  $\alpha_i^-$  must be zero, so their product is always zero:

$$\alpha_i^+ \alpha_i^- = 0. \quad (3.10)$$

Also notice that only those points outside the  $\epsilon$ -insensitive tube have  $\xi^+ > 0$  or  $\xi^- > 0$ . From Equation 3.9, it is worth mentioning that these outside points have a corresponding

$$\alpha_i^+ = C$$

or

$$\alpha_i^- = C.$$

Now consider a special case in which  $0 < \alpha_i^+ < C$ . It can be proved that:

$$\mu = y_i - \mathbf{w}^T \mathbf{x}_i - \epsilon \quad \text{if} \quad 0 < \alpha_i^+ < C. \quad (3.11)$$

In a similar way,

$$\mu = y_i - \mathbf{w}^T \mathbf{x}_i + \epsilon \quad \text{if} \quad 0 < \alpha_i^- < C. \quad (3.12)$$

### 3.2.4 Kernels

To capture more complicated landscapes of the unknown function, it is also desirable to make the SVR algorithm nonlinear. To achieve this, the input data from input space  $\mathcal{X}$  to the so-called feature space  $\mathcal{F}$  could be mapped:

$$\Phi : \mathcal{X} \rightarrow \mathcal{F}. \quad (3.13)$$

However, computing such  $\Phi$  explicitly can be prohibitive when the dimensionality of the input data is high, because the number of different monomial terms can grow rapidly as the dimension of the input space increases [Burges (1998)]. As noted at the end of section 3.2.2, it is only the dot product  $\mathbf{x}_i \cdot \mathbf{x}$  that is of interest in the SVR algorithm. Thus, it suffices to use a *kernel function* that only deals with the dot product in the algorithm instead, in order to avoid explicit computation of  $\Phi$ .

Those kernel functions  $k$ , which correspond to a dot product, obey the so-called Mercer's conditions. Some widely used kernels are:

- Homogeneous polynomial kernels

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^p \quad p \in \mathbb{N} \quad (3.14)$$

- Inhomogeneous polynomial kernels

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + c)^p \quad p \in \mathbb{N}, \quad c > 0 \quad (3.15)$$

- Gaussian

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp \left( -\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2} \right) \quad (3.16)$$

- Kriging

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp \left( -\sum_{i=1}^k \theta_k \left| \mathbf{x}_1^{(i)} - \mathbf{x}_2^{(i)} \right|^{p_k} \right) \quad (3.17)$$

Now the optimisation problem with the kernel concept included could be restated as:  
Maximise:

$$\begin{aligned} & \sum_{i=1}^k y_i (\alpha_i^+ - \alpha_i^-) - \epsilon \sum_{i=1}^k (\alpha_i^+ + \alpha_i^-) \\ & - \frac{1}{2} \sum_{i,j=1}^k (\alpha_i^+ - \alpha_i^-)(\alpha_j^+ - \alpha_j^-) k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \quad (3.18)$$

subject to:

$$\sum_{i=1}^k (\alpha_i^+ - \alpha_i^-) = 0 \quad \text{and} \quad \alpha_i^\pm \in [0, C].$$

### 3.2.5 Computing $\epsilon$ using $\nu$ -SVR

In many cases it is possible to estimate  $\epsilon$  based on the understanding of the engineering problem at hand. However, when such information is unavailable, value of  $\epsilon$  can still be calculated via the  $\nu$ -SVR method.

In the  $\nu$ -SVR, the constrained optimisation problem can be represented as:

$$\begin{aligned} & \text{minimise} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \left( \nu \epsilon + \sum_{i=1}^k (\xi_i^+ + \xi_i^-) \right) \\ & \text{subject to} \quad \begin{cases} y_i - \mathbf{w}^T \mathbf{x}_i - \mu \leq \epsilon + \xi_i^+ \\ \mathbf{w}^T \mathbf{x}_i + \mu - y_i \leq \epsilon + \xi_i^- \\ \xi_i^\pm \geq 0, \epsilon \geq 0. \end{cases} \end{aligned} \quad (3.19)$$

Following a similar procedure, (3.19) can be transformed into the dual optimisation problem:

Maximise:

$$\begin{aligned} & \sum_{i=1}^k y_i (\alpha_i^+ - \alpha_i^-) \\ & - \frac{1}{2} \sum_{i,j=1}^k (\alpha_i^+ - \alpha_i^-) (\alpha_j^+ - \alpha_j^-) k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \quad (3.20)$$

subject to:

$$\sum_{i=1}^k (\alpha_i^+ - \alpha_i^-) = 0, \quad \alpha_i^\pm \in [0, C]$$

and

$$\sum_{i=1}^k (\alpha_i^+ + \alpha_i^-) \leq C\nu.$$

To get the value of  $\epsilon$ , Figure 3.11 and 3.12 are equated and substitute into Figure 3.6

$$\begin{aligned} \epsilon = & \frac{1}{2} \left( y_m - y_n - \sum_{i=1}^k (\alpha_i^+ - \alpha_i^-) k(\mathbf{x}_i, \mathbf{x}_m) \right. \\ & \left. + \sum_{i=1}^k (\alpha_i^+ - \alpha_i^-) k(\mathbf{x}_i, \mathbf{x}_n) \right) \end{aligned} \quad (3.21)$$

if

$$0 < \alpha_m^+ < C \quad \text{and} \quad 0 < \alpha_n^- < C.$$

### 3.2.6 Tuning SVR parameters

The estimation accuracy of SVR depends on the setting of its parameters  $C$ ,  $\epsilon$  and kernel parameters. The problem of parameter selection is further complicated by the fact that SVM model complexity and hence its generalisation performance depends on all three parameters. In this work  $C$  and Gaussian kernel parameter  $\sigma$  are treated as user-defined inputs, and are manually tuned in order to get an optimised SVR performance.  $\epsilon$  is computed using  $\nu$ -SVR method. Many literature states that parameters should be selected based on the understanding of the engineering problem at hand. However, when such information is unavailable, it is difficult to tell the extent to which the parameters of a SVR model need to be tuned for the resulting surrogate model to be effective. Systematic ways of estimating SVR parameters could make an interesting research direction. Hsu et al. (2003) suggested that for a Gaussian kernel, best combination of  $C$  and  $\sigma$  can be selected by a grid search with exponentially growing sequences of  $C$  and  $\sigma$ . For example,  $C \in \{2^{-5}, 2^{-3}, \dots, 2^{13}, 2^{15}\}$  and  $\sigma \in \{2^{-15}, 2^{-13}, \dots, 2^1, 2^3\}$ . Each combination of parameter choices is checked using cross validation, and the parameters with best cross-validation accuracy are picked. The final model, which is used for testing and for classifying new data, is then trained on the whole training set using the selected parameters.

### 3.2.7 Implementation

To solve (3.18), the *quadprog* function in the MATLAB optimisation toolbox could be used. For example,

$$\mathbf{x} = \text{quadprog}(\mathbf{H}, \mathbf{f}, \mathbf{A}, \mathbf{b}, \mathbf{Aeq}, \mathbf{beq}, \mathbf{lb}, \mathbf{ub})$$

solves the quadratic programming problem:

$$\min_x \left( \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{f}^T \mathbf{x} \right) \quad (3.22)$$

subject to:

$$\begin{aligned} \mathbf{A} \cdot \mathbf{x} &\leq \mathbf{b}, \\ \mathbf{Aeq} \cdot \mathbf{x} &= \mathbf{beq}, \\ \text{and } \quad \mathbf{lb} &\leq \mathbf{x} \leq \mathbf{ub}, \end{aligned}$$

where  $\mathbf{H}$ ,  $\mathbf{A}$ , and  $\mathbf{Aeq}$  are matrices, and  $\mathbf{f}$ ,  $\mathbf{b}$ ,  $\mathbf{beq}$ ,  $\mathbf{lb}$ ,  $\mathbf{ub}$ , and  $\mathbf{x}$  are vectors. Figure 3.18 can be rewritten to fit it in the standard form (Equation 3.22) as:

maximise

$$\frac{1}{2} \begin{bmatrix} \alpha^+ \\ \alpha^- \end{bmatrix}^T \begin{bmatrix} \Psi & -\Psi \\ -\Psi & \Psi \end{bmatrix} \begin{bmatrix} \alpha^+ \\ \alpha^- \end{bmatrix} + \begin{bmatrix} \mathbf{1}^T \epsilon - \mathbf{y} \\ \mathbf{1}^T \epsilon + \mathbf{y} \end{bmatrix}^T \begin{bmatrix} \alpha^+ \\ \alpha^- \end{bmatrix} \quad (3.23)$$

subject to

$$\mathbf{1}^T \begin{bmatrix} \alpha^+ \\ \alpha^- \end{bmatrix} = 0$$

and

$$\alpha^\pm \in [0, C].$$

After  $\alpha^\pm$  are given by the quadratic programming solver, the complete form of the SVR predictor could be written out as

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^k (\alpha_i^+ - \alpha_i^-)(\mathbf{x}_i \cdot \mathbf{x}) + \mu \quad (3.24)$$

### 3.3 B-spline representation and derivatives

The inlet duct design problem that will be studied in Chapter 4 is a B-spline based model. Therefore, a brief review to B-splines representations and their derivatives is given in this section. B-splines are an efficient means of representing complex high-degree curves. In essence, they are seamless conjunctions of low degree Bézier curve segments. A B-spline curve is defined as follows:

$$\mathbf{C}(u) = \sum_{i=0}^n N_{i,p}(u) \mathbf{P}_i \quad (3.25)$$

where  $N_{i,p}(u)$  are the B-spline basis functions of degree  $p$  and  $\mathbf{P}_i$  are the control points. Given  $m$  real valued  $u_i$ , called “knots”, with  $u_0 \leq u_1 \leq \dots \leq u_{m-1}$ , the  $m - n + 1$  basis B-splines of degree  $n$  can be defined using the Cox-de Boor recursion formula [Piegl and Tiller (1997)].

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad (3.26)$$

$$N_{i,n}(u) = \frac{u - u_i}{u_{i+n} - u_i} N_{i,n-1}(u) + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_{i+1}} N_{i+1,n-1}(t).$$

It is sometimes necessary to find out the derivative of a B-spline for analysis. The derivative of a B-spline curve is:

$$\mathbf{C}'(u) = \sum_{i=0}^{n-1} N_{i,p-1}(u) \mathbf{Q}_i \quad (3.27)$$

where the  $\mathbf{Q}_i$ 's are defined as follows:

$$\mathbf{Q}_i = \frac{p}{u_{i+p+1} - u_{i+1}} (\mathbf{P}_{i+1} - \mathbf{P}_i). \quad (3.28)$$

Since the first derivative of a B-spline curve is another B-spline curve, derivatives of higher orders could be computed by recursively applying this technique.

The curvature  $k(u)$  at  $u$  can be computed as:

$$k(u) = |\mathbf{C}'(u) \times \mathbf{C}''(u)| / |\mathbf{C}'(u)|^3. \quad (3.29)$$

The radius of curvature  $R(u)$  is the reciprocal of curvature  $k(u)$ :

$$R(u) = \frac{1}{k(u)} \quad (3.30)$$

### 3.4 Geometry repair

After the knowledge surrogate is properly trained with a set of training data, typically obtained by evaluating the feasibility related penalty functions at a set of designs included in a sampling plan, the surrogate model can be used to predict the feasibility of untested geometries by giving a numerical value which can be regarded as a predictor of the feasibility of the unknown geometry. To draw a line between feasible and infeasible designs, a feasibility threshold  $p_{th}$  should be defined. The untested geometries which have a value higher than the threshold will be regarded as infeasible. Then, when an infeasible geometry is detected by the surrogate, an alternative set of design variables can be found automatically using an optimisation-based search over the surrogate, making an automatic geometry repair process possible.

The repair of a low feasibility geometry can be implemented by identifying the design alternative with the smallest possible repair alteration (SPRA). To be precise, the SPRA is defined as the set of design variable increments that will make the design feasible while keeping changes to a minimum, thus retaining the original design intent to the maximum [Sóbester and Keane (2006)]. The method will be further elaborated in Chapter 4.

In essence, the search for the SPRA is a single objective constrained optimisation problem:

$$\begin{aligned} &\text{minimise} && SPRA = \sum_{i=1}^n \sqrt{(x_i - x_{r_i})^2} \\ &\text{subject to} && \hat{f}(\mathbf{x}) \leq p_{th} \end{aligned} \quad (3.31)$$

where  $x_i$  is the  $i^{th}$  design variable of the unknown geometry  $\mathbf{x}$ ,  $x_{r_i}$  is the  $i^{th}$  design variable of the proposed repair alternative  $\mathbf{x}_r$  and  $\hat{f}(\mathbf{x})$  is the feasibility predictor of  $\mathbf{x}$ , given by the surrogate  $\hat{f}$ .



The choice of method to be employed for the search of a repair alternative is fairly open from the standpoint of computing budget because both the objective (the distance between the original design and repair alternatives) and the constraint (the SVR geometry quality predictor) are cheap to evaluate. However, if a simple gradient-based local search is employed, it is possible that the true nearest possible repair alternative is neglected if it hides behind a local hump of the surrogate. In this work, a variable resolution evolutionary strategy approach has been adapted to avoid such problems. The search starts with an initial global search, which covers the whole scope of the design space. If the initial global search is successful, further local searches are repetitively performed within the hyper-sphere which centred on the original design point with a radius equal to the distance between nearest feasible design found so far. In each repetition, or “generation” in the evolution strategies terminology, a series of offspring are obtained by using the full factorial sampling technique. If a whole generation fails to produce a better design alternative, the sampling density will be increased to allow a more intense search. After a successful round of search, the design variable sets that fulfil the quality constraint are listed, the one that is closest to the original design picked and used as the benchmark for the next round of search. The search terminates when either the optimised design reaches a pre-determined penalty value below which the design is deemed as “satisfactory” or the limit of computing budget is reached.

### 3.5 Summary

In this chapter, building blocks of the proposed knowledge-based geometry repair system are reviewed in detail. Knowledge can be gathered from physical judgement, simulation results and individual assessment of the geometry in question by a domain expert. After the knowledge is collected and normalised, it can be used to train a regression surrogate model. The resulting model represents the knowledge at hand. It is used for the subsequent search for the SPRA, in order to find a robust and close substitute. In Section 3.3 mathematics of B-spline representation and analysis are reviewed, which will be used in the next chapter.

As a summary of this chapter, a flowchart of the proposed process is shown in Figure 3.2. In the preparation phase, expert knowledge is translated into penalty functions. At the same time, a sampling plan is generated across the design space. Each sample is evaluated by referencing to the penalty functions and/or CFD/FEA codes or individual assessment by an expert. Evaluations are stored in a database, which is used to construct a knowledge surrogate. The surrogate is tuned to best represent the training data before it is used to predict unknown geometry feasibility and search for the SPRA if the new unknown geometry is predicted to be infeasible. Of course there is no certain way to determine the feasibility threshold, therefore the algorithm generates and uses a set of feasibility thresholds, based on which a sequence of repaired geometries is generated.

The designer can review the repaired geometries in the order of increasing or decreasing distance from the original design and may choose one to form the repaired geometry. When new feasible geometries are determined, the designer may probably be interested in its true penalties/physical properties. These values can be calculated as before, and the data obtained can be used to update the existing database.

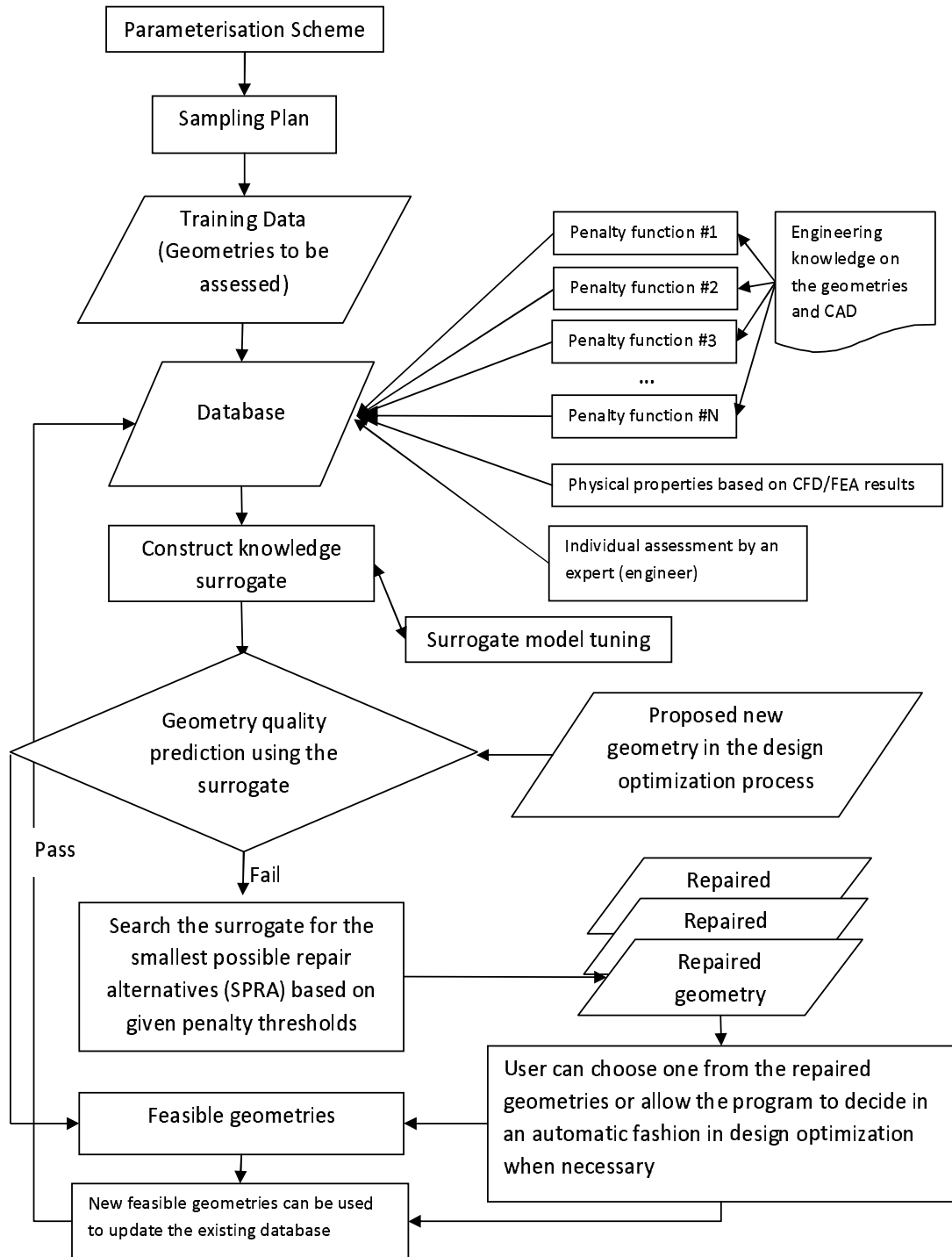


FIGURE 3.2: Methodology flowchart for setting up the repair system and the repair process



## Chapter 4

# Case Study: Geometry Repair in the Aero-engine Intake Design

In this chapter, the geometry repair methodology proposed in Chapter 3 is applied to an aero-engine intake geometry model design case. An overview of the aero-engine intake design problem is presented first in Section 4.1. Section 4.2 presents the setup of the intake model and the geometry parameterisation scheme. Practical physical-based engineering knowledge are summarised and converted into penalty functions in Section 4.3. The repair system is set up in Section 4.4 and the effectiveness of the repair system is tested over a few individual geometries, the movement of the repair suggestions being illustrated on a repair path chart. Later in Chapter 4, different regression models are used for modelling the knowledge, and a comparison of the penalty prediction landscape and the repair results generated by using different surrogate methods are given in Section 4.5. Finally a graphical user interface that visualise SPRAs and assists the selection of an optimal design by a designer is presented in Section 4.6.

### 4.1 Overview of the intake design problem

Aero-engine intake ducts are designed to meet a number of criteria to ensure that an aircraft engine is properly supplied with steady, high quality airflow under different operational conditions from steady flight to maneuvering. The duct should capture a portion of the incoming free stream airflow and transform it into suitable flow conditions before feeding it into the entrance of the engine fan or compressor [Seddon and Goldsmith (1985)].

As the air is brought from free stream to the compressor face, the flow may be distorted by the inlet duct. At the compressor face, one portion of the flow may have a higher velocity or higher pressure than another portion. The flow may be swirling, or some

section of the boundary layer may be thicker than another section because of the inlet shape. The rotor blades of the compressor move in circles around the central shaft. As the blades encounter distorted inlet flow, the flow conditions around the blade change very quickly. The changing flow conditions can cause flow separation in the compressor, a compressor stall, undesirable vibration of rotating blade rows or the possibility of engine surge (reverse airflow) and can cause structural problems for the compressor blades. A good inlet must produce low distortion [NASA (2009)].

As mentioned above, flow separation and distortion under cross wind or tail wind conditions should be minimised. While at cruise conditions, it is also important to minimise the drag by optimising the design of the intake together with other components. The design of a turbine-powered aircraft engine intake is a demanding task due to its multi-objective and multi-disciplinary nature. Major objectives in the design process include minimising drag, compressor entrance pressure distortions, weight, cost and maximising pressure recovery [Sobester (2007)]. The design of a modern inlet is always a compromise between these major objectives.

The advent of supersonic aircraft raised new challenges on the efficiency of inlet since the total pressure recovery significantly influences the overall engine performance. Annular intakes with bullet centre-bodies (e.g. MiG-21) and normal shock intakes (e.g. F-16 Falcon) became favourable designs for fighter aircraft. Other important factors that influence the design decision include capture area of the intake, effect of flow distortion, noise reduction. For civil aircraft, non-integrated podded engine installations are dominant, raising another challenge of the design and optimisation of nacelle and its integration with the engine.

## 4.2 The intake model and its parameterisation

Here a simple 2-dimensional parameterized inlet model was built as a test case, as shown in Figure 4.1. The external shape of the airframe, the position of the engine and rear bulkhead are fixed. The intake shape is entirely dependent upon its centre axis (the dashed curve in the graph), which is a B-spline defined by seven control points (C1–C7, as noted in the graph). The horizontal positions of these control points are held constant. The vertical positions of C1 and C2 are kept the same so that the entrance of the intake stays horizontal. Additionally, the vertical positions of C6 and C7 are on the engine centreline to ensure that the exit of the intake is level and connects smoothly with the engine. The cross-sectional area of the intake duct equals that of the engine face and is held constant along the duct centre axis. The vertical positions of other points are left free as design variables. Horizontal positions, acceptable vertical position ranges and corresponding design variables for each control point are listed in Table 4.1.

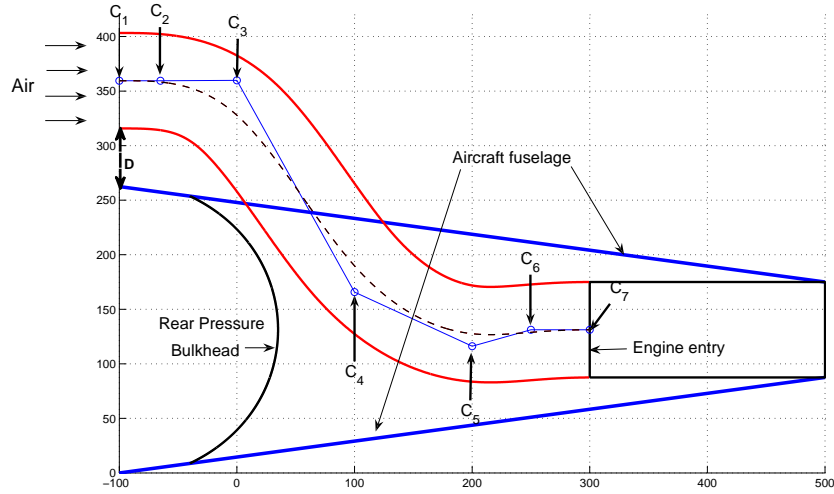


FIGURE 4.1: A simplified 2D intake model

The design variable set  $\mathbf{x}$  comprises four design variables in the test case:

$$\mathbf{x} = \{x_1, x_2, x_3, x_4\}.$$

For example, the design variable set  $\mathbf{x} = \{359.5, 359.9, 165.8, 116.1\}$  will result in the design shown in Figure 4.1.

Control Point	Horizontal position	Vertical range	Design variable
$C_1$	-100	245 – 385	$x_1$
$C_2$	-65	same as $C_1$	same as $C_1$
$C_3$	0	175 – 455	$x_2$
$C_4$	100	70 – 350	$x_3$
$C_5$	200	35 – 315	$x_4$
$C_6$	250	131.25	
$C_7$	300	131.25	

TABLE 4.1: List of the horizontal positions, acceptable vertical position ranges and corresponding design variables for each control point

### 4.3 Physics based knowledge of the intake design and its representation

As mentioned in Section 3.1, engineering knowledge in the form of explicit rules can be transformed into penalty functions in order to make them optimisable objectives. In this section, the idea is illustrated by three examples.

### 4.3.1 Consideration on the vertical distance between the fuselage and intake

First of all, an intake position penalty function  $P_1$  is related to the vertical distance  $D$  between the aircraft fuselage (upper boundary) and the intake lower boundary at the air intake entrance position, as show in Figure 4.1. A positive  $D$  corresponds to a protruding intake design, which will increase the aerodynamic drag. On the other hand, negative distance will result in an intake design where the entrance is partially submerged into the fuselage. Since the air capture area is reduced, the capture/throat area ratio is reduced, which is undesirable from an intake aerodynamics point of view. Such engineering considerations are incorporated into  $P_1$ , in which both unfavourable scenarios receive a penalty.  $P_1$  is considered as having the form

$$P_1 = \begin{cases} D^{3/2} & \text{if } D \geq 0 \\ -100D & \text{if } D < 0 \end{cases} \quad (4.1)$$

The reason an exponential function was chosen when  $D \geq 0$  and a linear function when  $D < 0$  was to test the learning ability of support vector regression with different penalty functions. Since the airframe external shape is fixed and the vertical position of the intake entrance is defined by the design variable  $x_1$ ,  $D$  is solely dependent upon  $x_1$ . Here  $x_1$  and  $D$  are related by

$$D = x_1 - 306.25, \quad (4.2)$$

so that  $P_1$  can be rewritten as a function of the design variables

$$P_1 = P_1(\mathbf{x}) = \begin{cases} (x_1 - 306.25)^{3/2} & \text{if } x_1 - 306.25 \geq 0 \\ -100(x_1 - 306.25) & \text{if } x_1 - 306.25 < 0. \end{cases} \quad (4.3)$$

### 4.3.2 Consideration on the curvature limit of the intake duct

A second penalty function  $P_2$  is related to the curvature of the intake duct. It can be seen that certain design variable combinations can render the overall shape of the intake duct convoluted. From an aerodynamic engineer's point of view, designs with sharp bends are unfavourable because the airflow can separate and cause distorted pressure fields on the engine face. Furthermore, too high a curvature of the centre line can cause a loop in its upper and lower offset curves and render the design impractical. Such a failed design is shown in Figure 4.2, in which the design variables are set to be  $\mathbf{x} = \{360, 300, 50, 250\}$ .

Therefore, to represent such engineering concerns, a penalty function  $P_2$  is set up to penalise geometries with excessive curvature. The curvature  $k(u)$  and the radius of

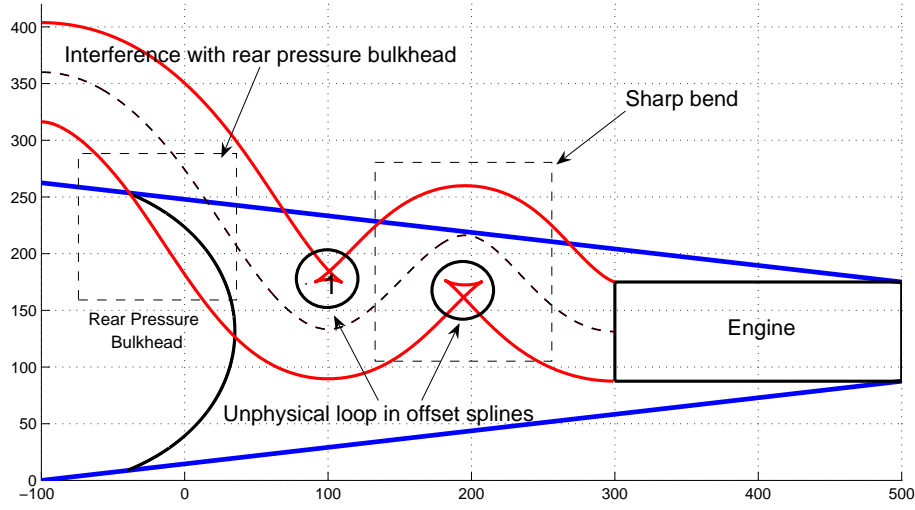


FIGURE 4.2: Failed design example due to high curvature

curvature  $R(u)$  of the centre B-spline axis can be computed by Equation 3.29 and Equation 3.30 respectively.

A loop will occur in the offset curve if  $R(u)$  is less than the radius of the engine face, which is 43.75 in this intake design case.  $P_2$  is set up as a sum of  $R(u)$ s whenever its value is less than 43.75, i.e.

$$P_2 = \sum_{u=0}^1 R(u), \forall R(u) < 43.75. \quad (4.4)$$

### 4.3.3 Interference with the rear pressure bulkhead

It is noticed that those duct designs which interfere with the rear pressure bulkhead are undesirable since precious space in the fuselage is occupied it involves moving the bulkhead forward. Such an unfavourable geometry is illustrated in Figure 4.3. A penalty function  $P_3$  is set up to penalise interference between the two parts.

$$P_3 = \text{Total Area of Interference} \quad (4.5)$$

A more severe interference will incur a higher penalty value of  $P_3$ . So far, three penalty functions have been set up for three explicit rules, each of which represents some form of engineering knowledge. Compared to direct consultation with a human expert or evaluation of CFD codes, explicit rules are relatively cheap to calculate and thus more favourable in the process of generating training data. However, in contrast to the 4 design variables in the test case, many more design variables may exist in real engineering



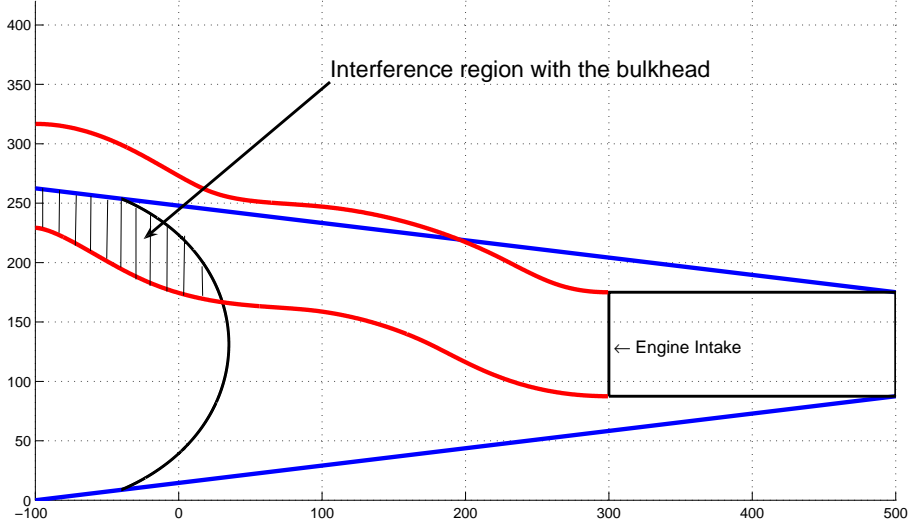


FIGURE 4.3: A faulty geometry due to structure interference

applications with underlying interactions that the designer may be unaware of. In the region of the search space where these explicit rules cannot reach, a CFD analysis could be set up, and it may be possible to consult human experts to get information about the deficiencies of the geometry. Human expert consultation and CFD analysis will result in case-based knowledge, which can be used in the training of a surrogate.

## 4.4 Repair result

### 4.4.1 A simple case

Following the procedure that has been described in the previous sessions, it is possible to predict the quality of the geometry and find an SPRA for faulty geometries. To test the idea of geometry repair, the author began with the single penalty function  $P_3$  which penalises the interference between the rear pressure bulkhead and intake, and a pre-determined feasibility threshold value  $p_{th} = 1500$ . A faulty design whose corresponding design variable set is  $\mathbf{x} = [0.2, 0.1, 0.5, 0.5]$  is presented in Figure 4.3. It is obvious that the interference between the bulkhead and intake is severe.

100 training points  $\mathbf{x}_i, i = 1, 2, \dots, 100$  were chosen by the Latin Hypercube sampling method. For each  $\mathbf{x}_i$ ,  $y_i = P_3(\mathbf{x}_i)$ . The SVR geometry quality predictor  $\hat{f}(\mathbf{x})$  was trained on these  $(\mathbf{x}_i, y_i)$  pairs, with  $i = 1, 2, \dots, 100$ . The predicted penalty for this design was found to be  $\hat{f}(\mathbf{x}) = 2.7874 \times 10^4$ . The search started globally over the whole design region. A Pareto front was determined based on the initial search result (the

circled points in Figure 4.4). The point which lay below and is closest to the feasibility

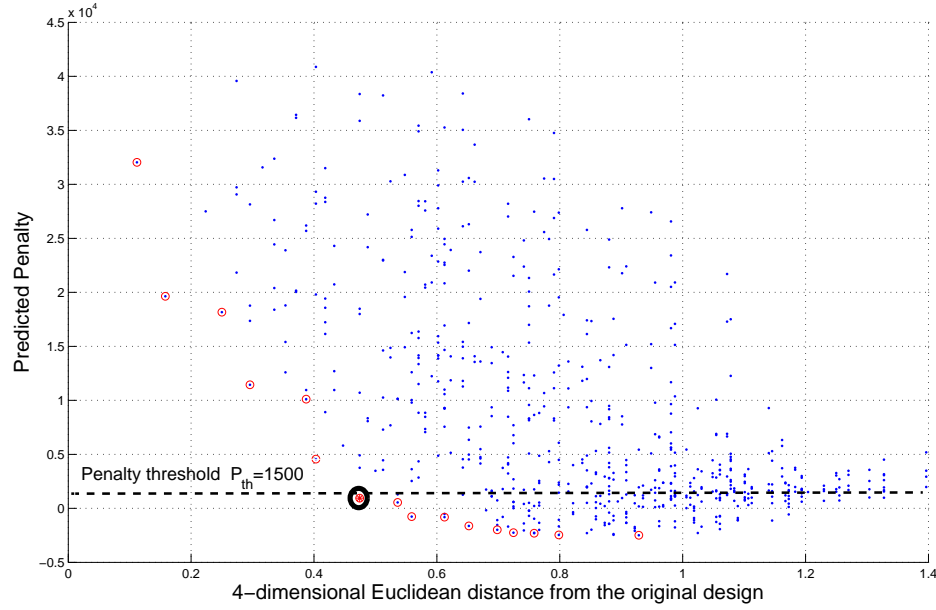


FIGURE 4.4: Initial global search result with its Pareto front being marked by circle, and the initial choice being marked by a bold circle

threshold line (black dashed line) was chosen as the current best point in the search process (bold circled point in Figure 4.4). It was predicted by the SVR predictor that the chosen point had a penalty of 940 and its Euclidean distance from the original design was 0.4743.

After the successful initial global search (with a sample size of  $11^4$ ), optimisation continued by repetitively performing local searches within the hyper-sphere centred at the original design point with a radius equal to the distance from the best point found so far. The sampling points outside the hyper-sphere were discarded to save computing budget. This strategy proved to be able to improve the search result in each of the four consecutive rounds before it converged at  $\mathbf{x} = [0.2650, 0.4900, 0.6950, 0.5650]$  with  $\hat{f}(\mathbf{x}) = 1440$ . The geometry corresponding to the optimised  $\mathbf{x}$  is shown in Figure 4.5. Comparing Figure 4.3 and Figure 4.5, it can be seen that in the optimised design the interference between the rear pressure bulkhead and the engine intake had been significantly reduced, while the aft part of the intake was subject only to minor changes. This indicates that the design can be made feasible by the repair process while the original design intent is retained.

#### 4.4.2 Repair of a design candidate with multiple flaws

The proposed geometry repair method proves feasible in the simple case study in Section 4.4.1. In this section, the author investigated into the application of the proposed

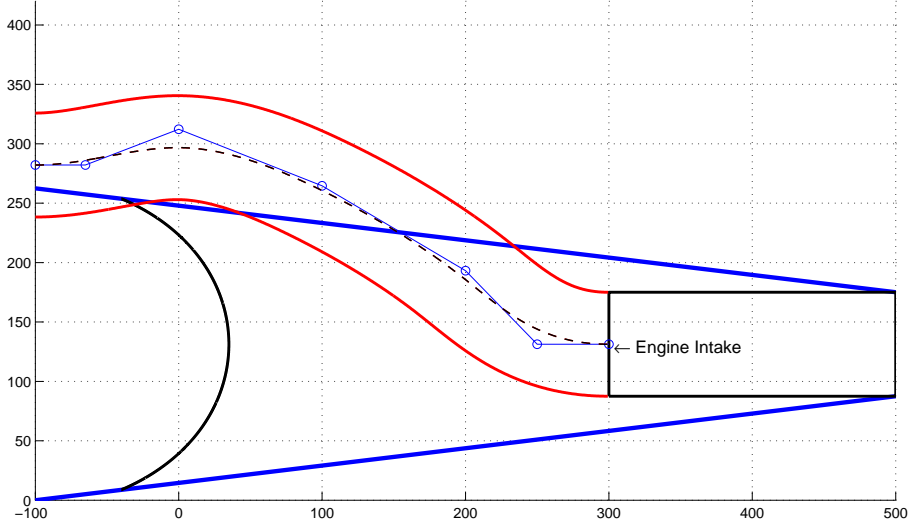


FIGURE 4.5: Suggested repair alternative of the original design, see Figure 4.3 for comparison

method on a design candidate with multiple design flaws. Such a flawed design is presented in Figure 4.6 with its design vector

$$\mathbf{x}_o = [0.45, 0.35, 0.05, 0.65].$$

The front part of the intake is slightly submerged in the fuselage and the duct itself is snaky and interferes with the bulkhead.

Three penalty functions are combined to form  $P = P_1 + P_2 + P_3$ . The same set of training points  $\mathbf{x}_i$  were used but this time  $y_i = P(\mathbf{x}_i)$ . An SVR surrogate  $\hat{f}(\mathbf{x})$  was trained on the data and was used for the optimisation. In this case, 10 different  $p_{th}$  values were used:  $p_{th} = 0, \hat{f}(\mathbf{x}_s)/10, 2\hat{f}(\mathbf{x}_s)/10, \dots, 9\hat{f}(\mathbf{x}_s)/10$  which led to 10 different repair suggestions and leave the final option to the engineer. The repair alternatives are presented in Table 4.2, (notice that the predicted penalty for the original design  $\mathbf{x}_o$  is  $\hat{f}(\mathbf{x}_s) = 5642$ ):

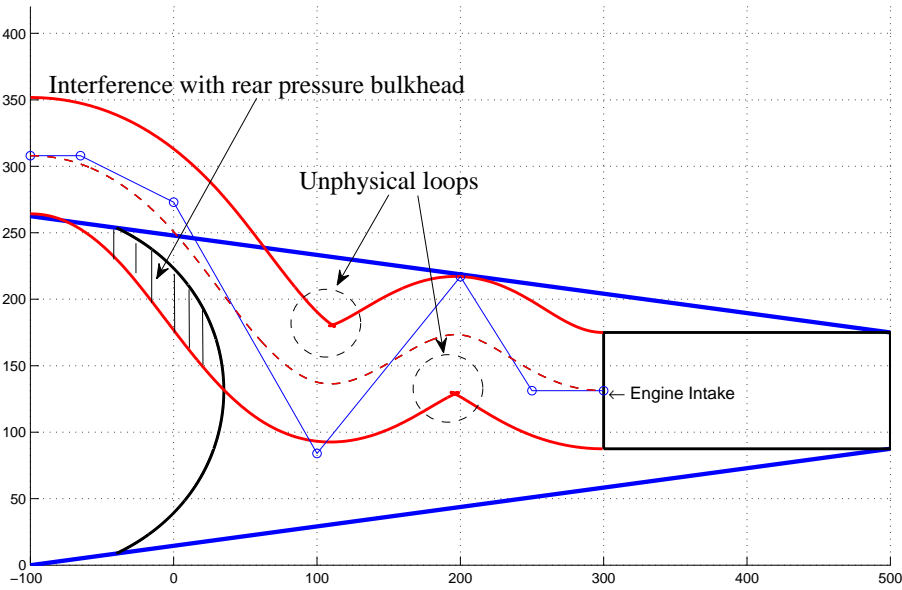


FIGURE 4.6: A design candidate with multiple flaws

Table 4.2: Design alternatives based on different feasibility threshold levels

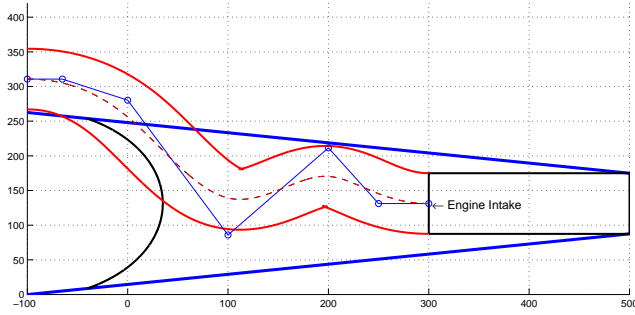
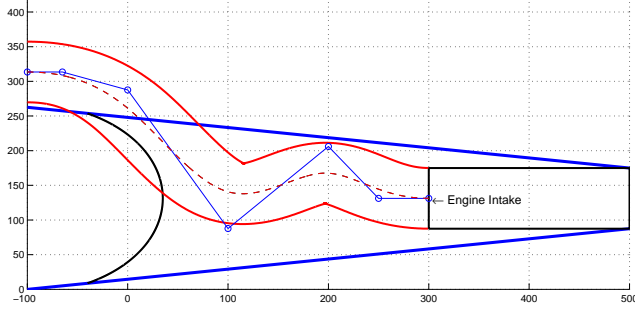
$p_{th}$	$\mathbf{x}$	Repair alternative suggestion
5053	$[0.4696, 0.3761, 0.0565, 0.6304]$	
4492	$[0.4892, 0.4022, 0.0631, 0.6108]$	

Table 4.2: (continued)

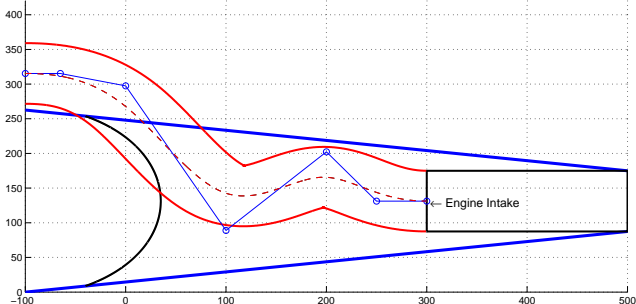
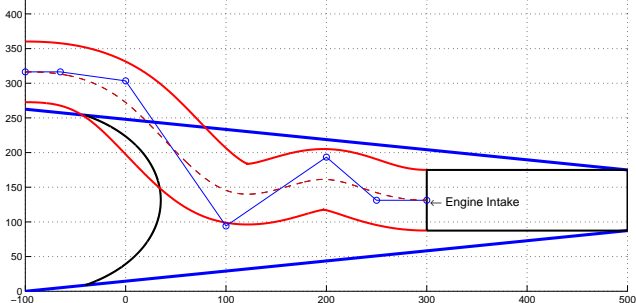
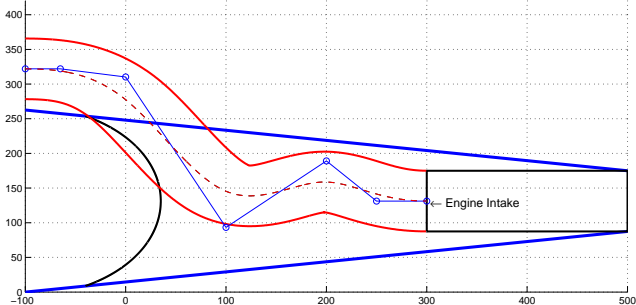
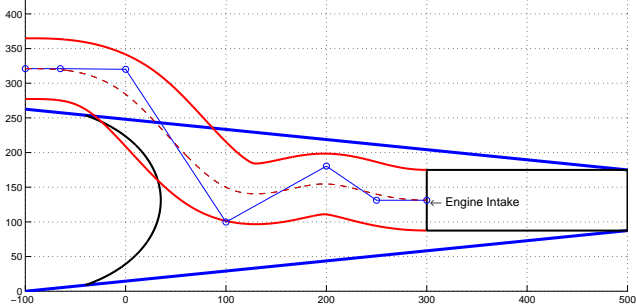
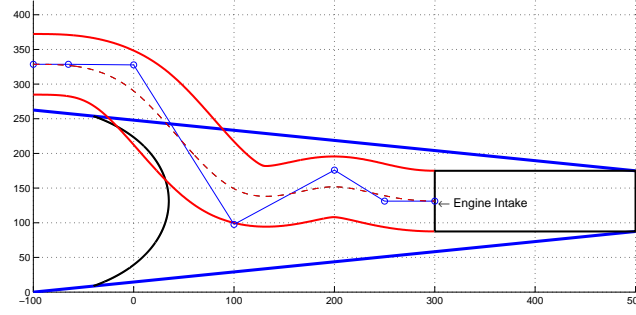
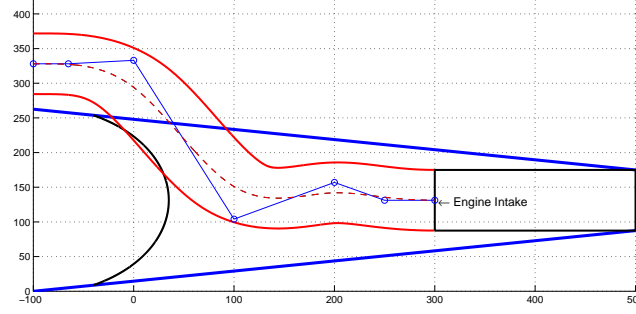
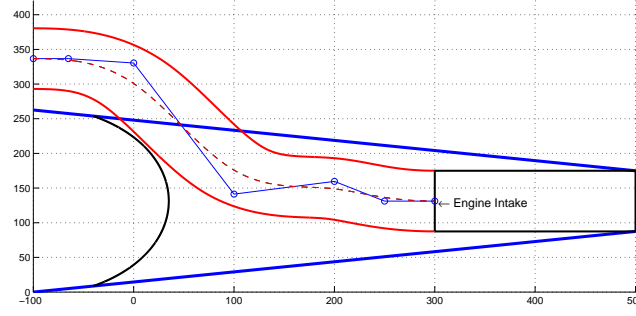
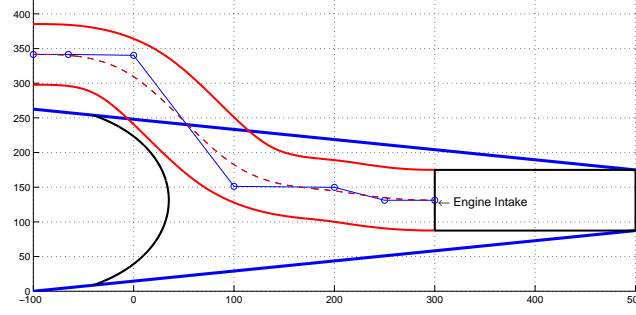
$p_{th}$	$\mathbf{x}$	Repair alternative suggestion
3930	[0.5025, 0.4375, 0.0675, 0.5975]	
3369	[0.5103, 0.4585, 0.0862, 0.5656]	
2807	[0.5496, 0.4828, 0.0832, 0.5504]	
2246	[0.5434, 0.5181, 0.1060, 0.5193]	

Table 4.2: (continued)

$p_{th}$	$\mathbf{x}$	Repair alternative suggestion
1684	$[0.5966, 0.5454, 0.0989, 0.5034]$	
1123	$[0.5929, 0.5644, 0.1215, 0.4356]$	
561	$[0.6548, 0.5548, 0.2548, 0.4452]$	
0	$[0.6900, 0.5900, 0.2900, 0.4100]$	

It can be seen that the repair suggestions  $\mathbf{x}_r$  gradually change from very similar to the original design to a “near perfect design” as the feasibility threshold  $p_{th}$  gradually decreases. The part exhibiting high curvature is gradually smoothed, while at the same time the interference with fuselage and rear bulkhead is gradually reduced. The engineer

can thus choose one of these optimised design alternatives in a way that best balances between the desire to retain the original design intent and the “health” of the geometry.

#### 4.4.3 Illustration of the repair path

To better illustrate the repair path on the predicted penalty function landscape, the code is slightly modified so that two of the design variables were fixed, allowing the other two variables to vary and to be optimised. The optimisation problem is thus reduced to a two dimensional one. The original design labelled No.1 in Figure 4.7 is  $\mathbf{x} = [0.4, 0.2, 0.2, 0.9]$ . In this case, the second and fourth design variable ( $x_2$  and  $x_4$ ) are allowed to vary. The contour plot on the left of Figure 4.7 shows the predicted penalty function landscape with regard to  $x_2$  and  $x_4$ . It could be clearly observed from Figure 4.7 that as the penalty threshold decreases, the suggested repair alternative moves away from the original design, and towards the area in the penalty landscape where the predicted penalty is low.

### 4.5 A comparison of the penalty prediction landscape and the repair result between different surrogate methods

The penalty prediction landscape can be constructed from the same sampling points by different surrogate modelling methods. The experiments presented here are aimed at determining the possible extent of these differences in the landscape and the repair result. In this section, the author constructed the penalty prediction functions by three other surrogate modelling methods: RBF network with thin plate spline base (Figure 4.8), Gaussian base (Figure 4.9) and inverse multi-quadric base (Figure 4.10). A detailed review of the methods and how the hyper-parameter  $\sigma$  is chosen can be found in Section 2.3.2. These prediction functions are presented against the SVR prediction (Figure 4.11) as well as the original penalty function (Figure 4.10). To visualise these four-dimensional functions, a “nested tile plot” [Forrester et al. (2008)] is used. In a nest tile plot, two variables are selected to plot against each other on each tile; then a matrix of such tiles could be generated, where the row and column of a tile determines the values of the remaining two variables.

To compare the quality of the repair results based on the different surrogates, ten random faulty designs were chosen from the design space (Table 4.3) and were repaired using the same repair algorithm as described in Section 3.4, each time based on one of the four different knowledge bases, which are represented by the different penalty prediction models (Figure 4.8 to 4.11).

The ten faulty design cases are listed in Table 4.3.

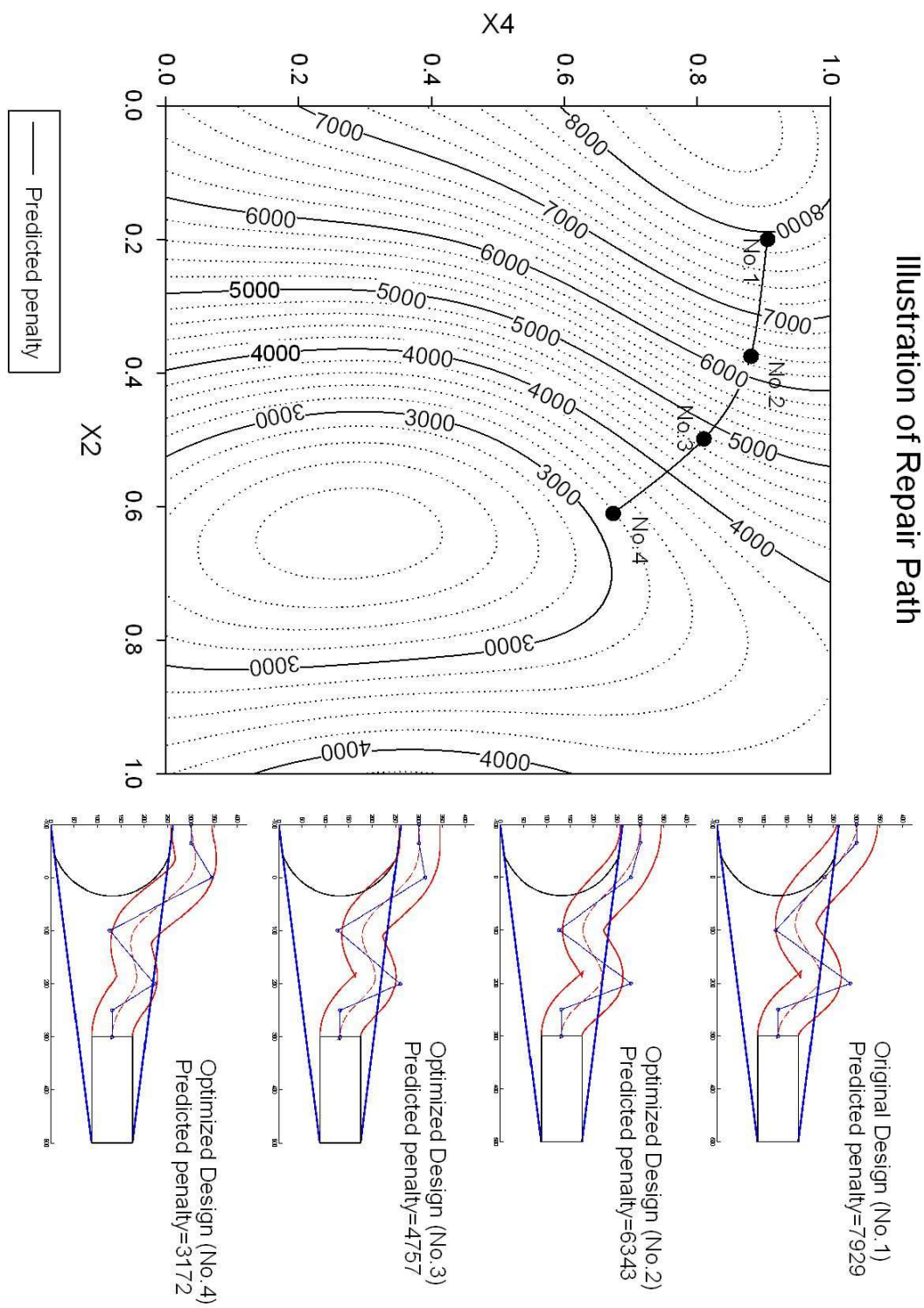


FIGURE 4.7: Illustration of a repair path



TABLE 4.3: Variables of ten faulty design cases

Case	Variable#1	Variable#2	Variable#3	Variable#3
1	0.005	0.555	0.255	0.645
2	0.185	0.015	0.295	0.135
3	0.015	0.275	0.775	0.305
4	0.055	0.835	0.635	0.885
5	0.065	0.205	0.755	0.745
6	0.025	0.125	0.105	0.145
7	0.125	0.455	0.185	0.975
8	0.035	0.315	0.265	0.195
9	0.045	0.855	0.405	0.005
10	0.075	0.065	0.205	0.765

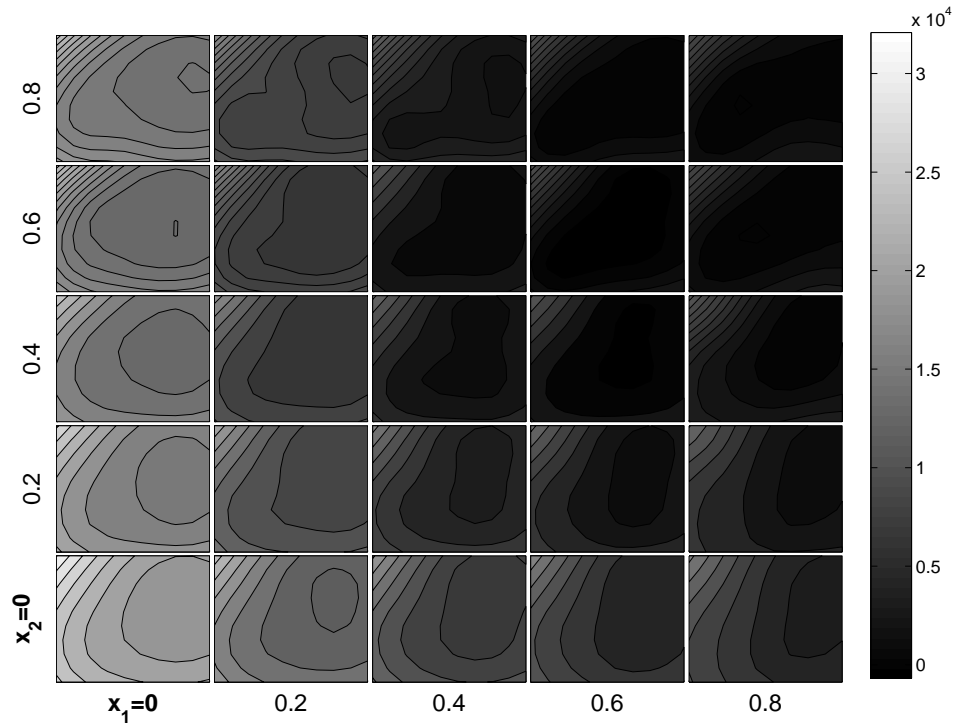


FIGURE 4.8: RBF with thin plate base prediction landscape

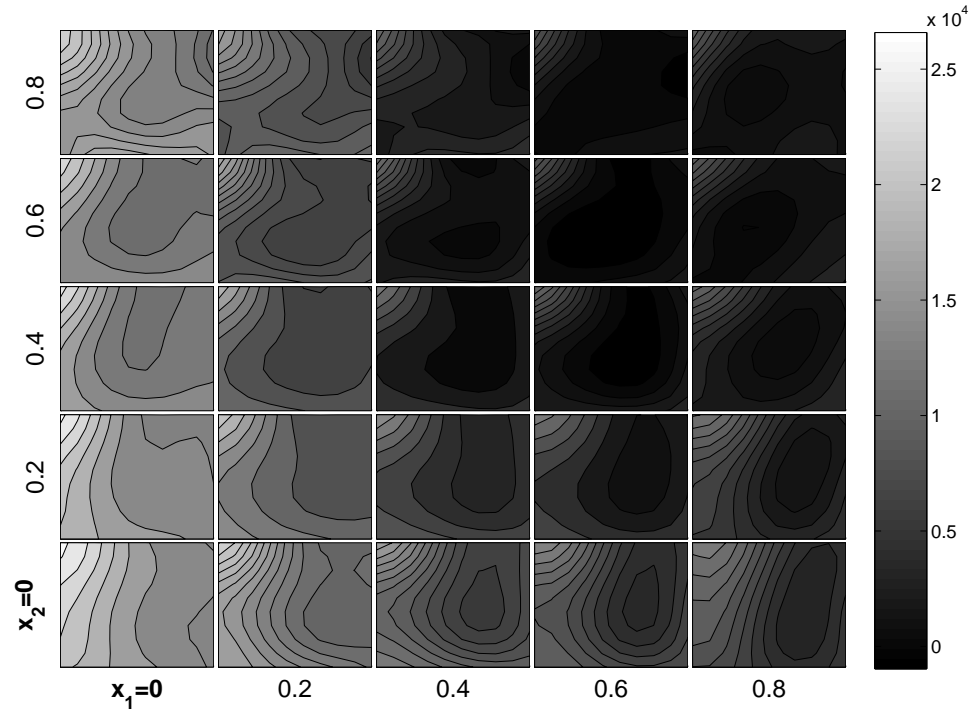


FIGURE 4.9: RBF with Gaussian base prediction landscape

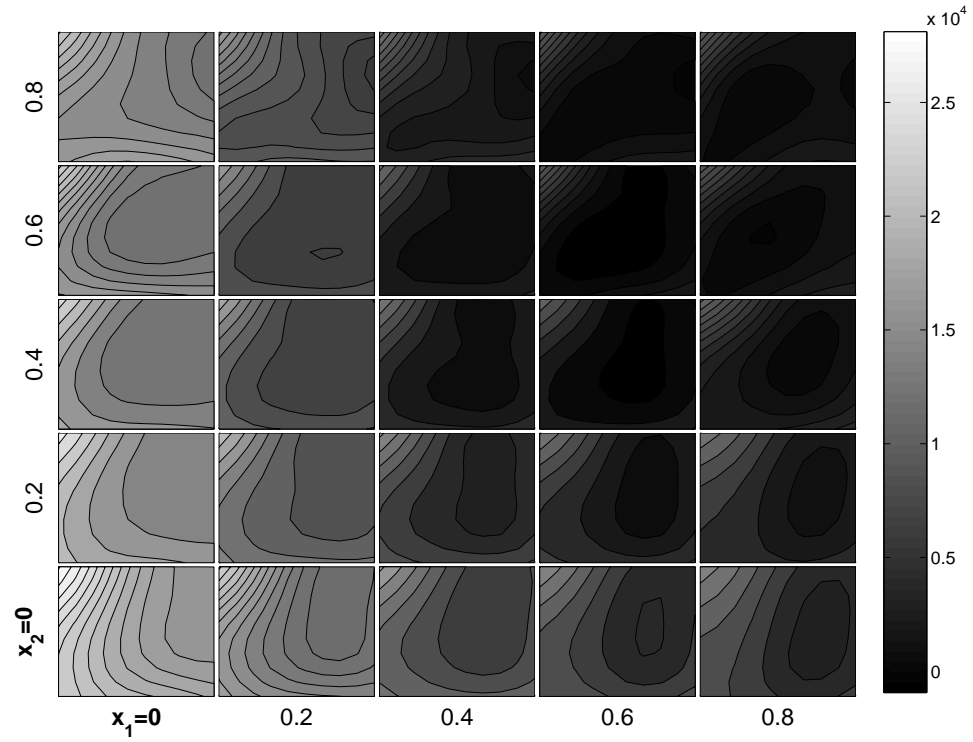


FIGURE 4.10: RBF with inverse multi-quadric base prediction landscape

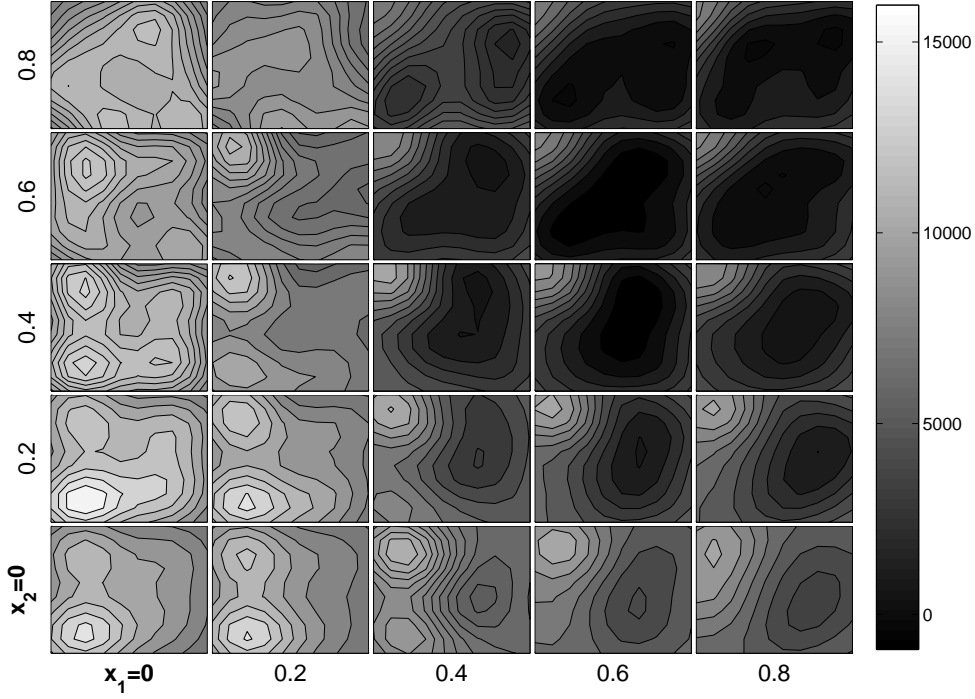


FIGURE 4.11: SVR prediction landscape

The analytical penalty function (Figure 4.12) is also used as a knowledge base as a control.

For each of the unrepaired faulty designs, ten repair alternatives are suggested by the repair algorithm. For each of the repair alternatives, the Euclidian distances to the unrepaired design ( $D_{min}$ ) are plotted on the x axis against the predicted penalty values (y axis). The data obtained by using the five knowledge bases are plotted on the same graph in order to compare with each other. The graphs for each unrepaired designs are presented in Figure 4.13, 4.15, 4.17, 4.19, 4.21, 4.23, 4.25, 4.27, 4.29 and 4.31.

The differences of penalty values of the repairs based on the four different surrogates and that based on the true knowledge model could be used as a measure of the repair quality of the different knowledge bases. The smaller the differences are, the closer the surrogates are to the true knowledge base. Formally, for a certain type of surrogate, a quality indicator  $Q$  would be expressed as:

$$Q = \sum_{i=1}^n |\text{Predicted penalty} - \text{True penalty}| \quad (4.6)$$

where  $n$  is the total number of the repair alternatives.

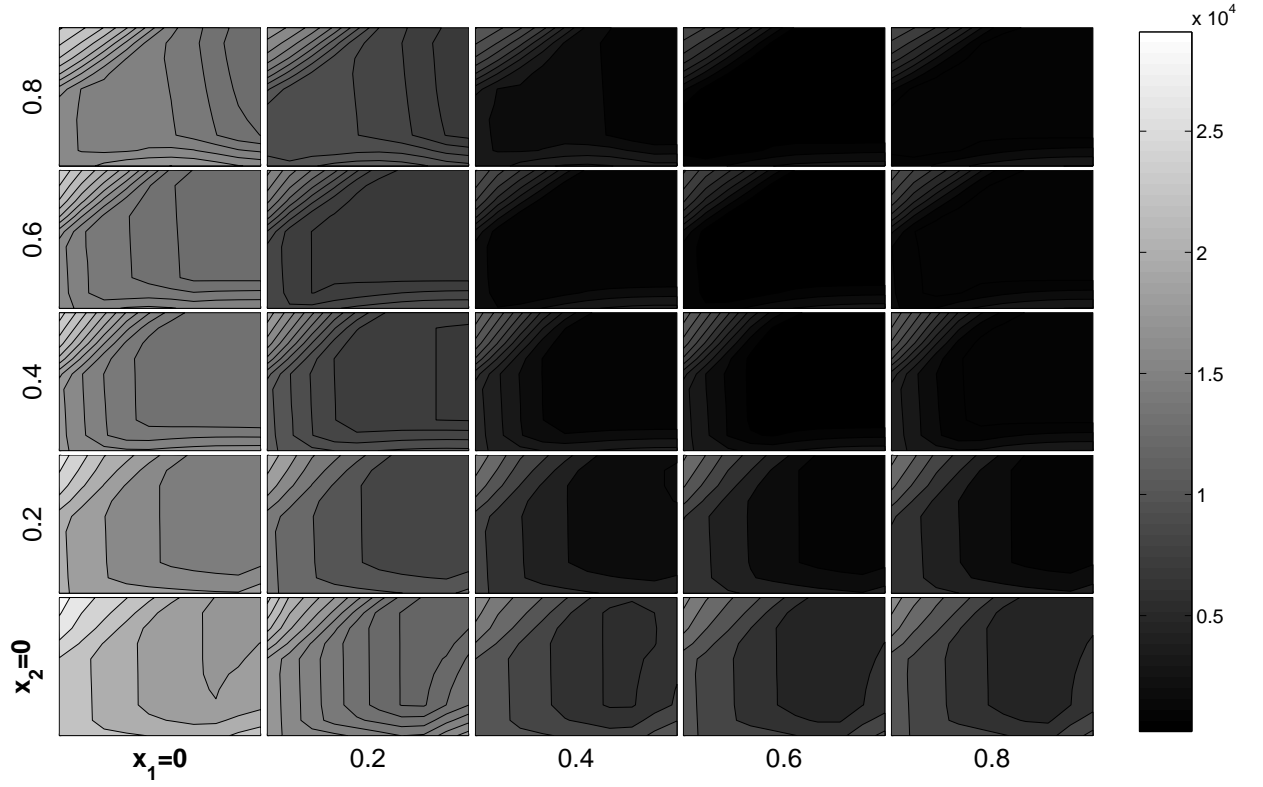


FIGURE 4.12: Analytical penalty function landscape

Therefore, it is necessary to calculate the true penalty values of the suggested repair alternatives and compare them with the corresponding predicted penalty values. The true penalty values of the suggested repair alternatives are determined by using the suggested repair variables as the input of the true penalty function. The values of  $D_{min}$  are plotted against the true penalty values in the same way as they are plotted against the predicted penalty values. The results are presented in Figure 4.14, 4.16, 4.18, 4.20, 4.22, 4.24, 4.26, 4.28, 4.30 and 4.32.

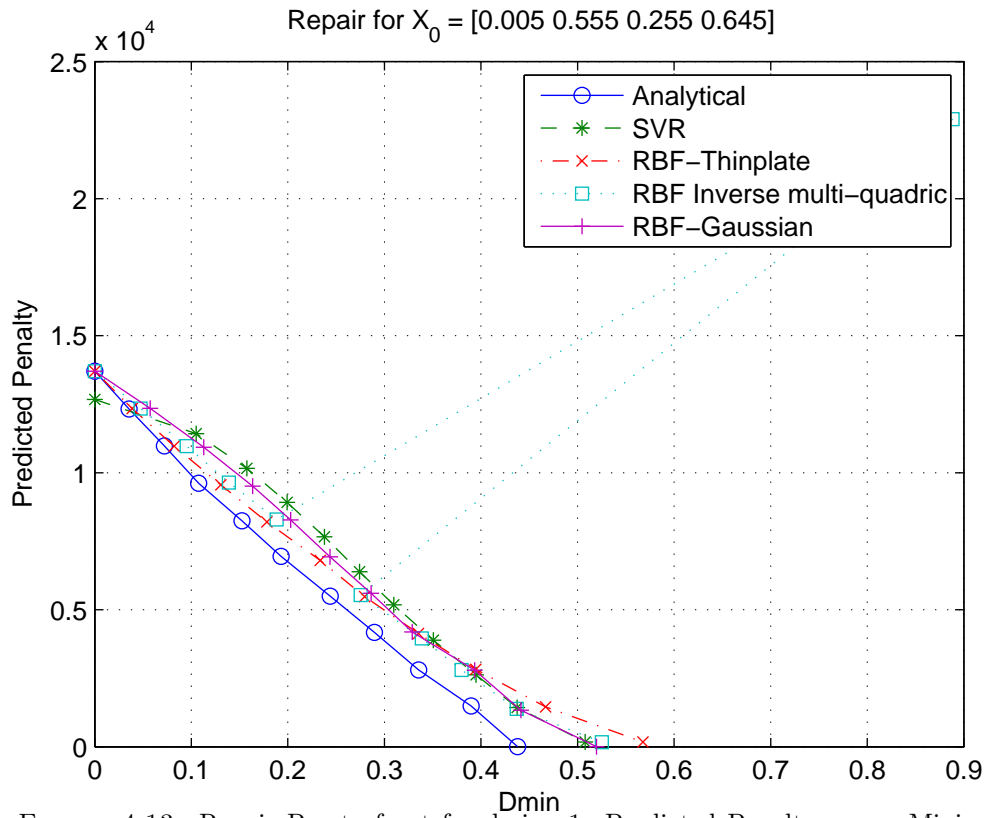


FIGURE 4.13: Repair Pareto front for design 1: Predicted Penalty versus Minimum distance

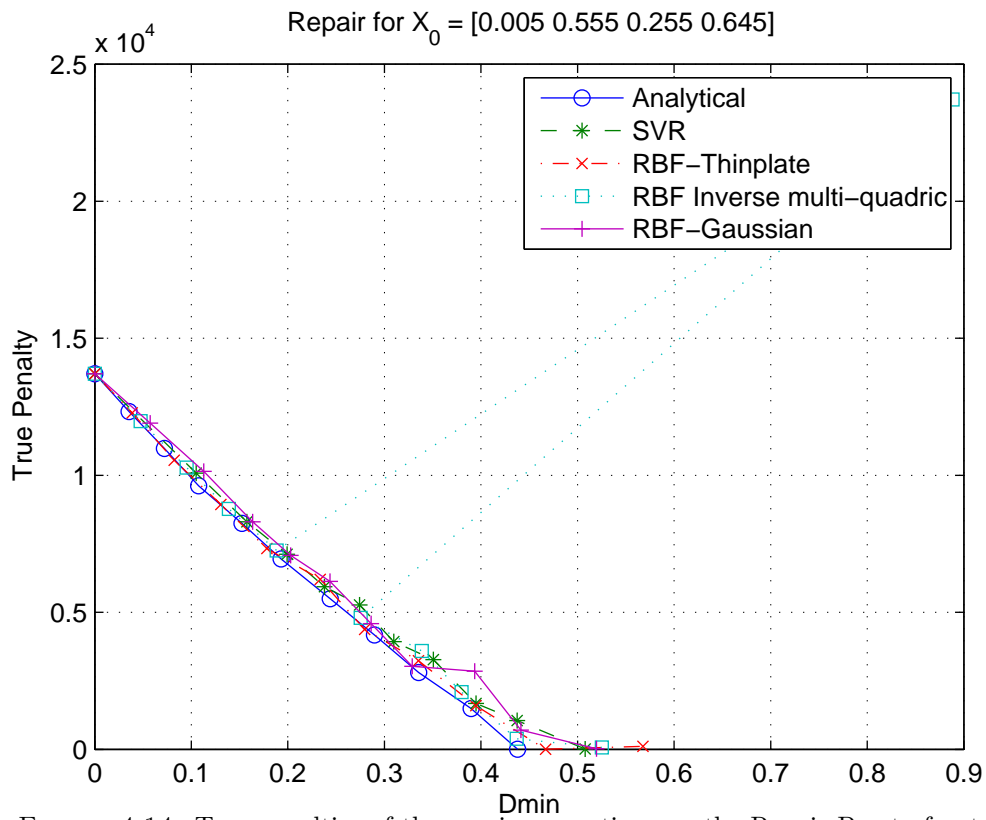


FIGURE 4.14: True penalties of the repair suggestions on the Repair Pareto front for design 1

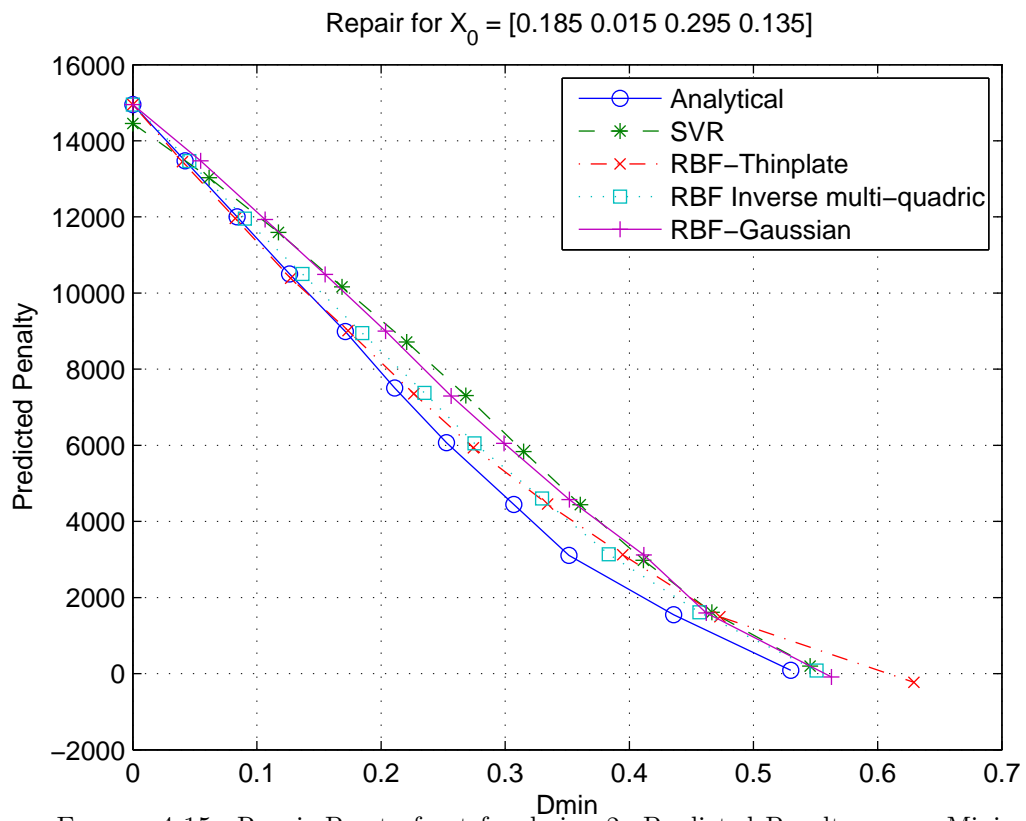


FIGURE 4.15: Repair Pareto front for design 2: Predicted Penalty versus Minimum distance

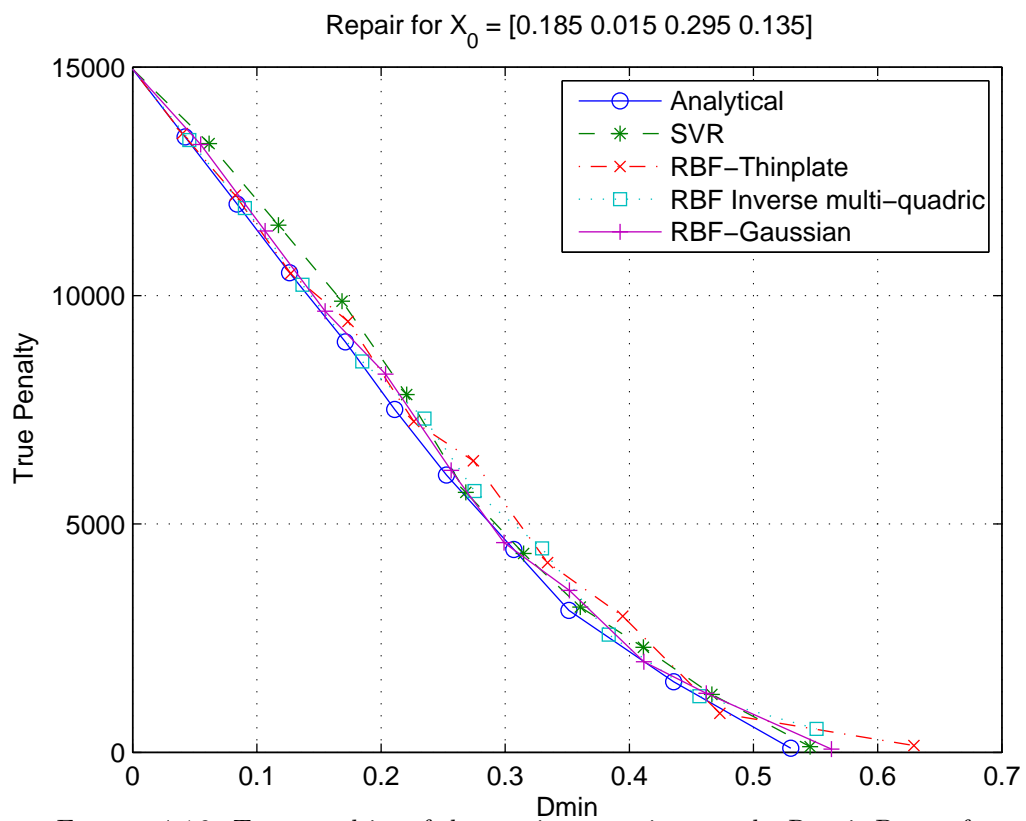


FIGURE 4.16: True penalties of the repair suggestions on the Repair Pareto front for design 2

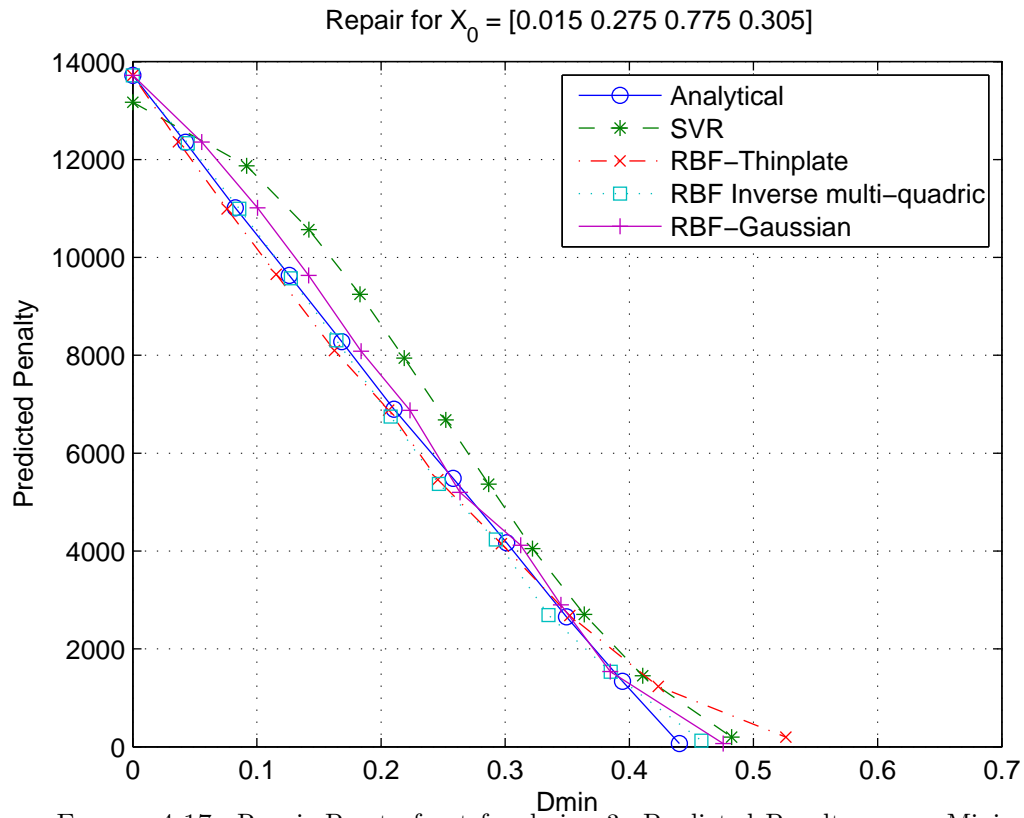


FIGURE 4.17: Repair Pareto front for design 3: Predicted Penalty versus Minimum distance

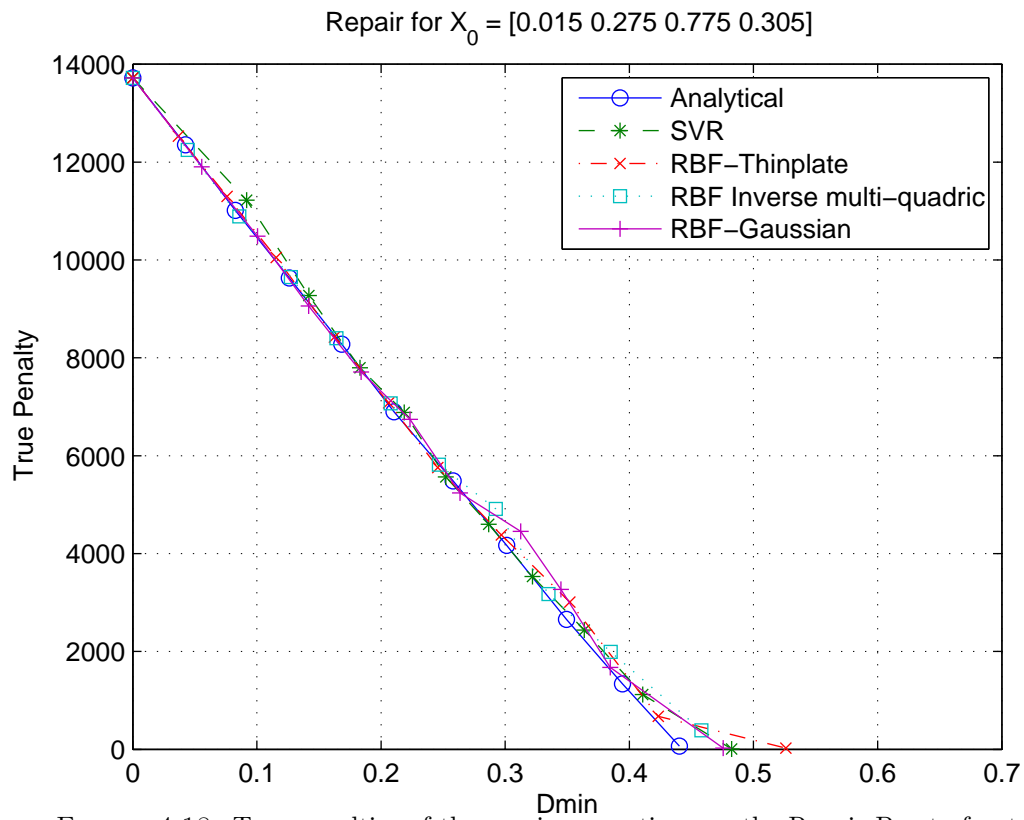


FIGURE 4.18: True penalties of the repair suggestions on the Repair Pareto front for design 3

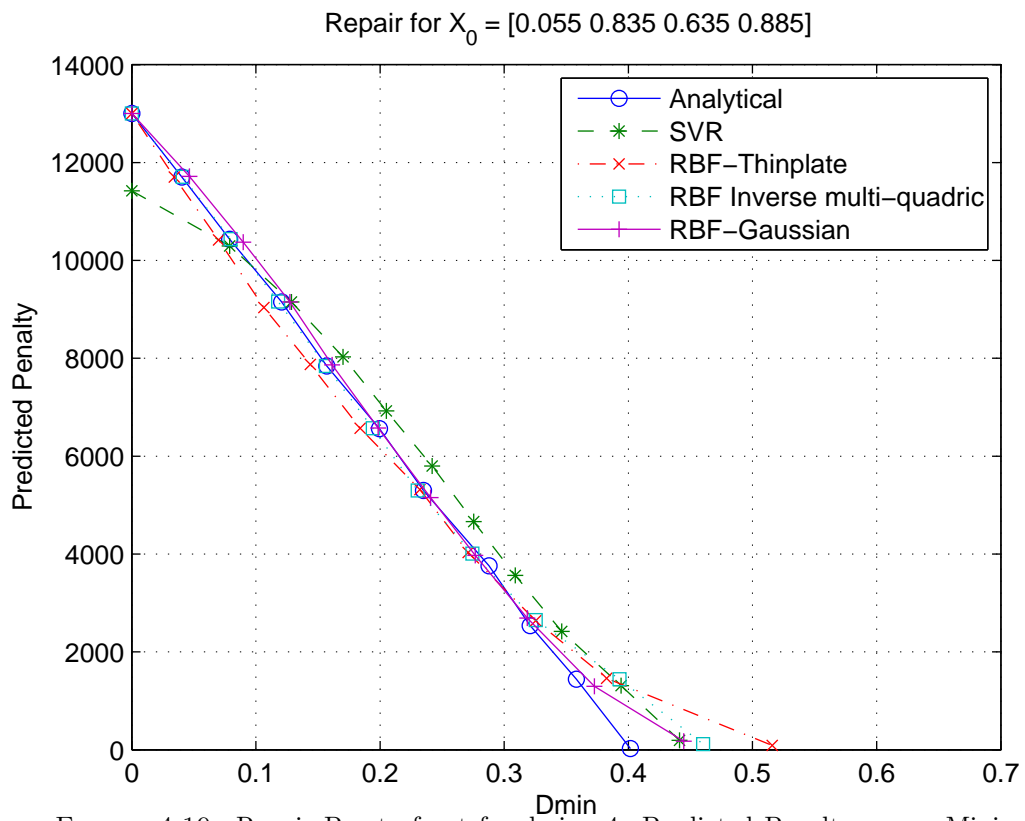


FIGURE 4.19: Repair Pareto front for design 4: Predicted Penalty versus Minimum distance

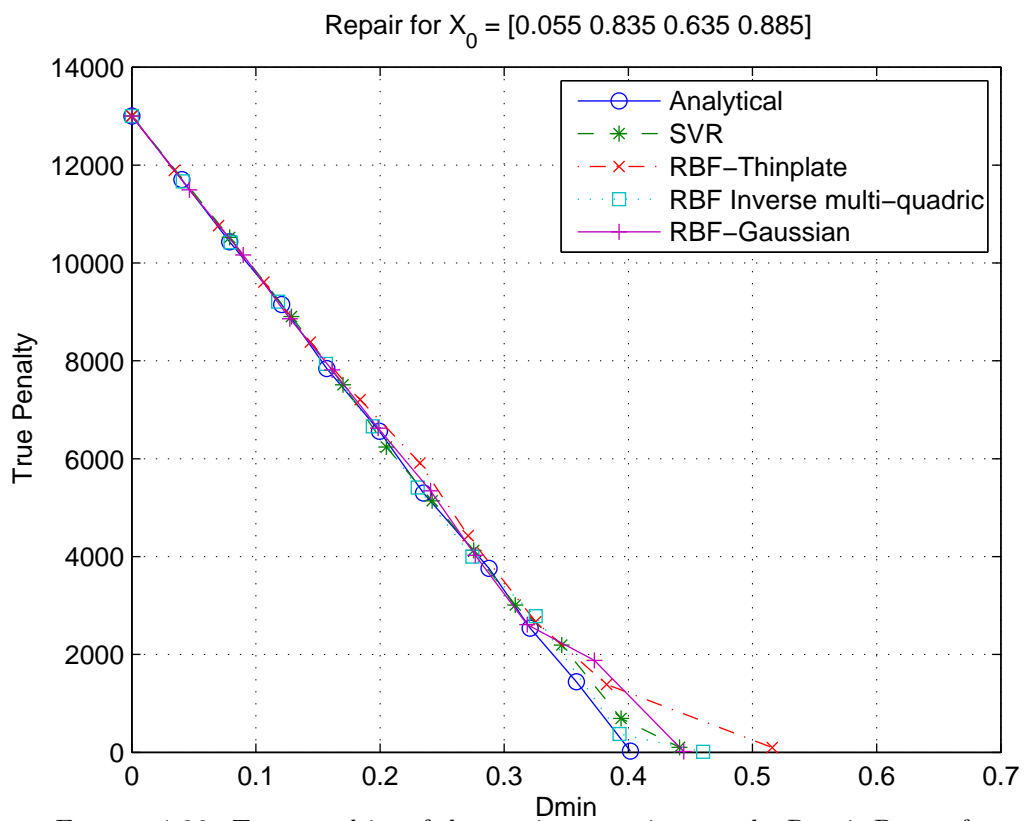


FIGURE 4.20: True penalties of the repair suggestions on the Repair Pareto front for design 4



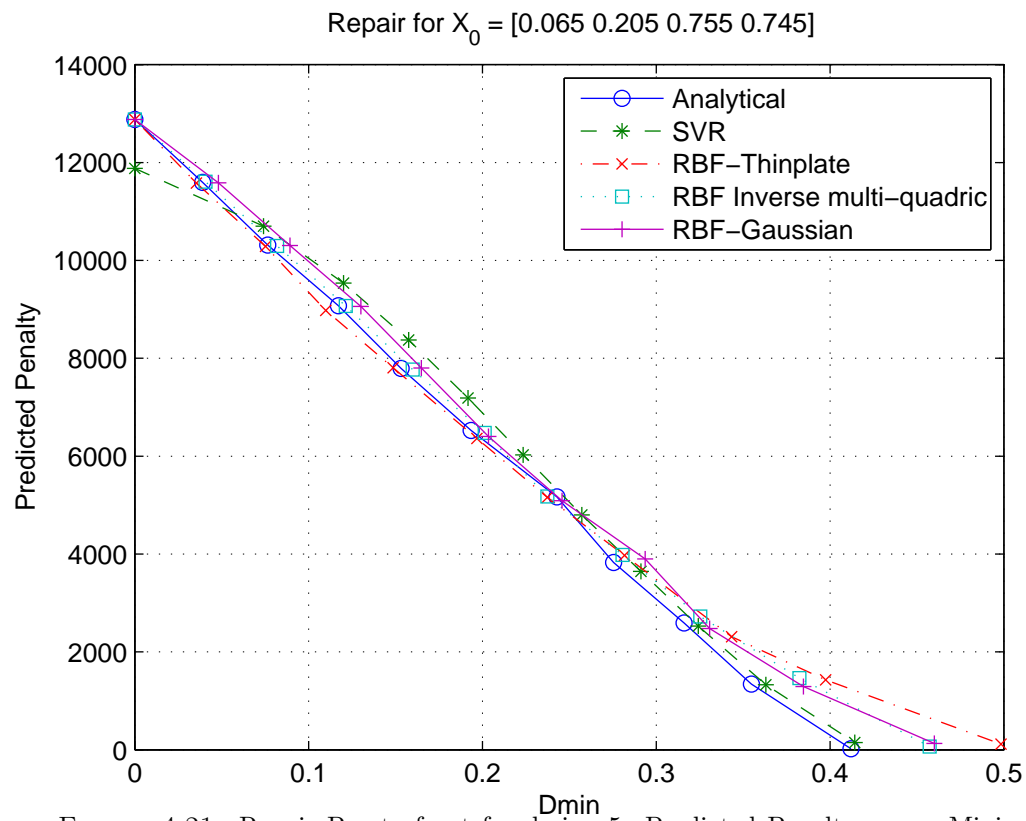


FIGURE 4.21: Repair Pareto front for design 5: Predicted Penalty versus Minimum distance

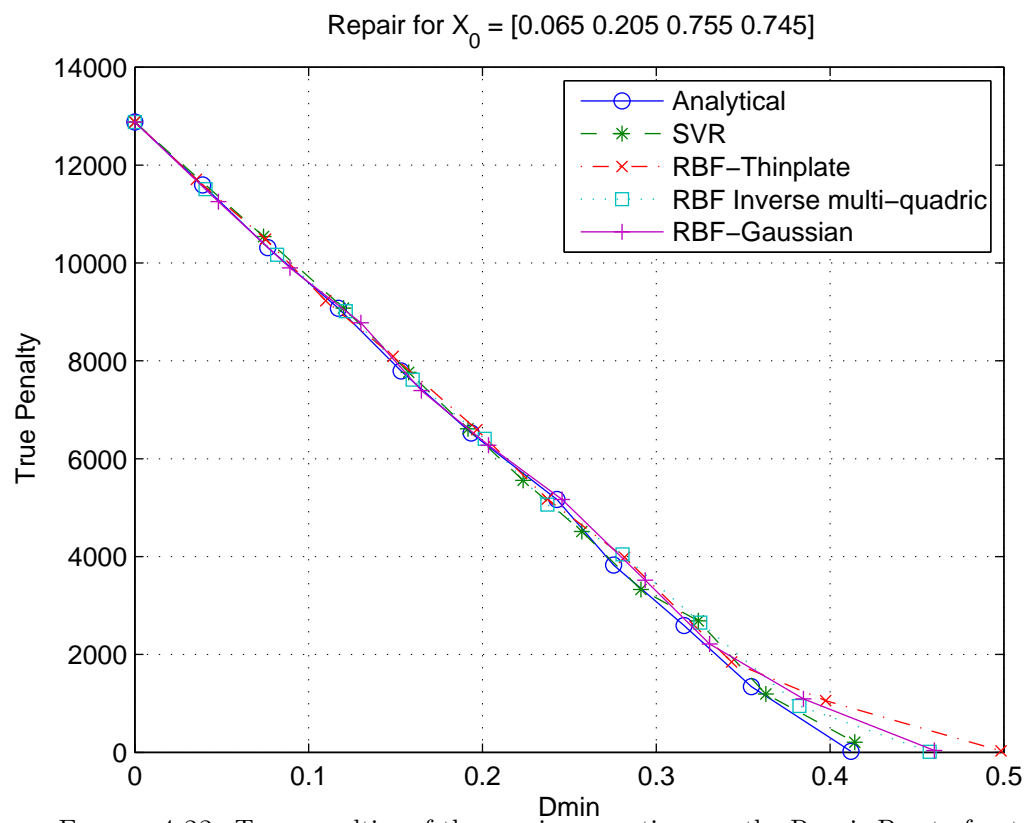


FIGURE 4.22: True penalties of the repair suggestions on the Repair Pareto front for design 5

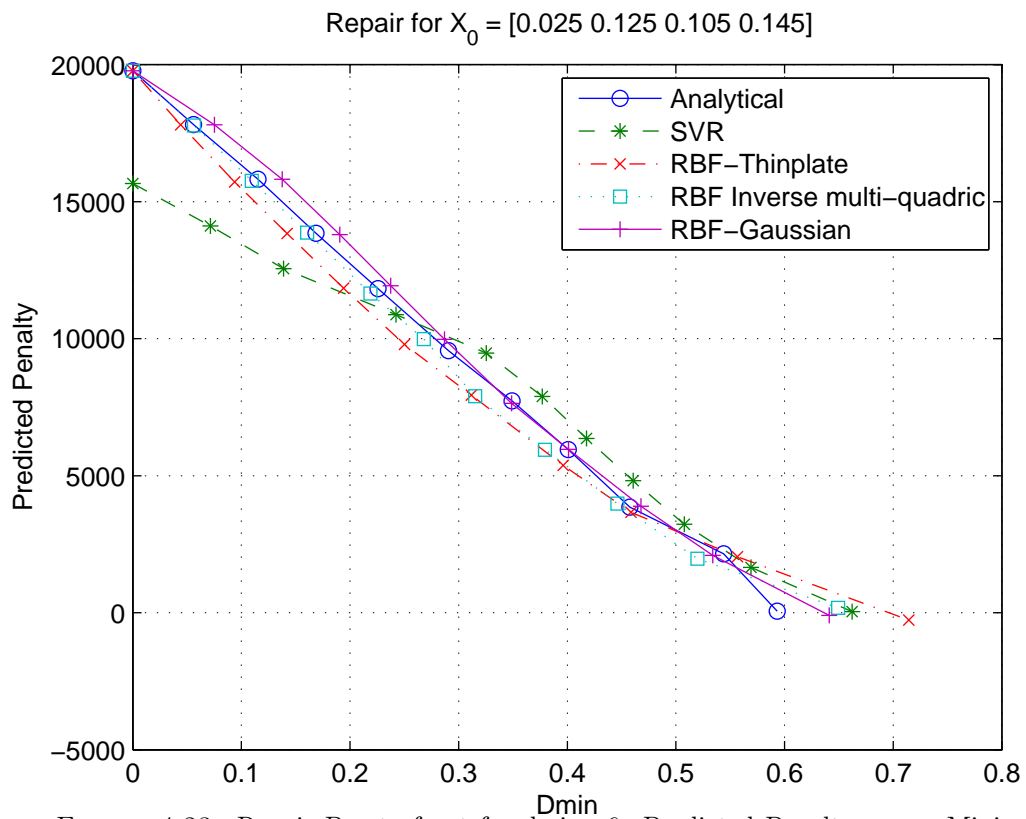


FIGURE 4.23: Repair Pareto front for design 6: Predicted Penalty versus Minimum distance

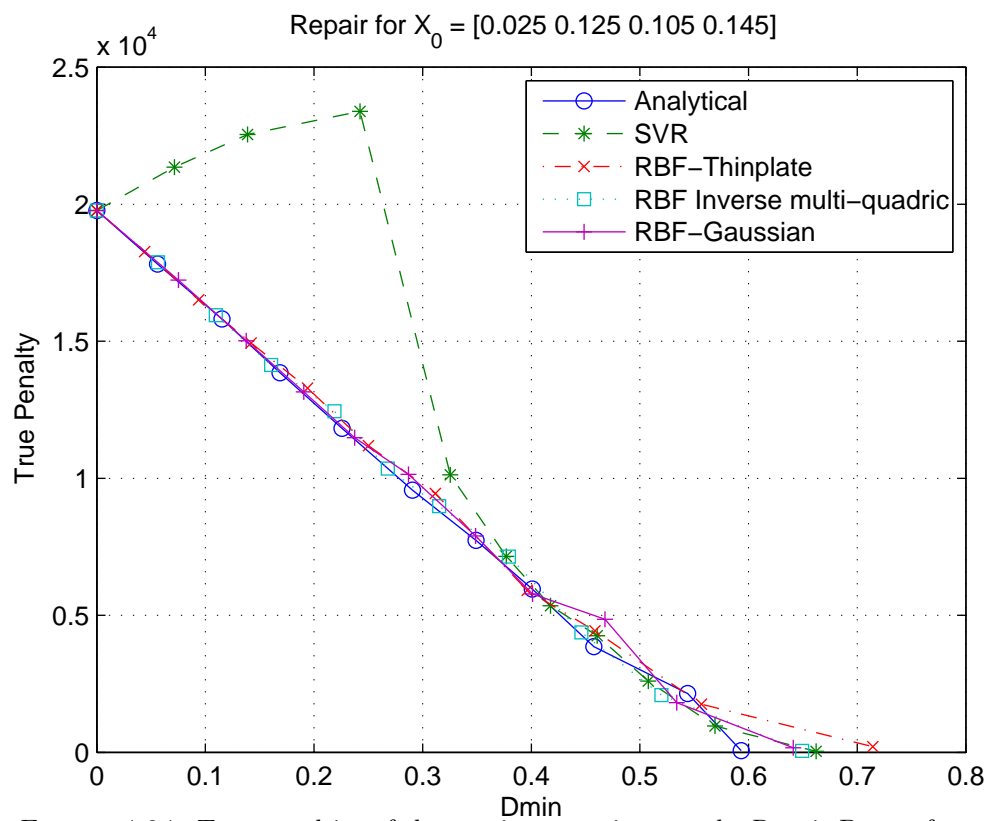


FIGURE 4.24: True penalties of the repair suggestions on the Repair Pareto front for design 6

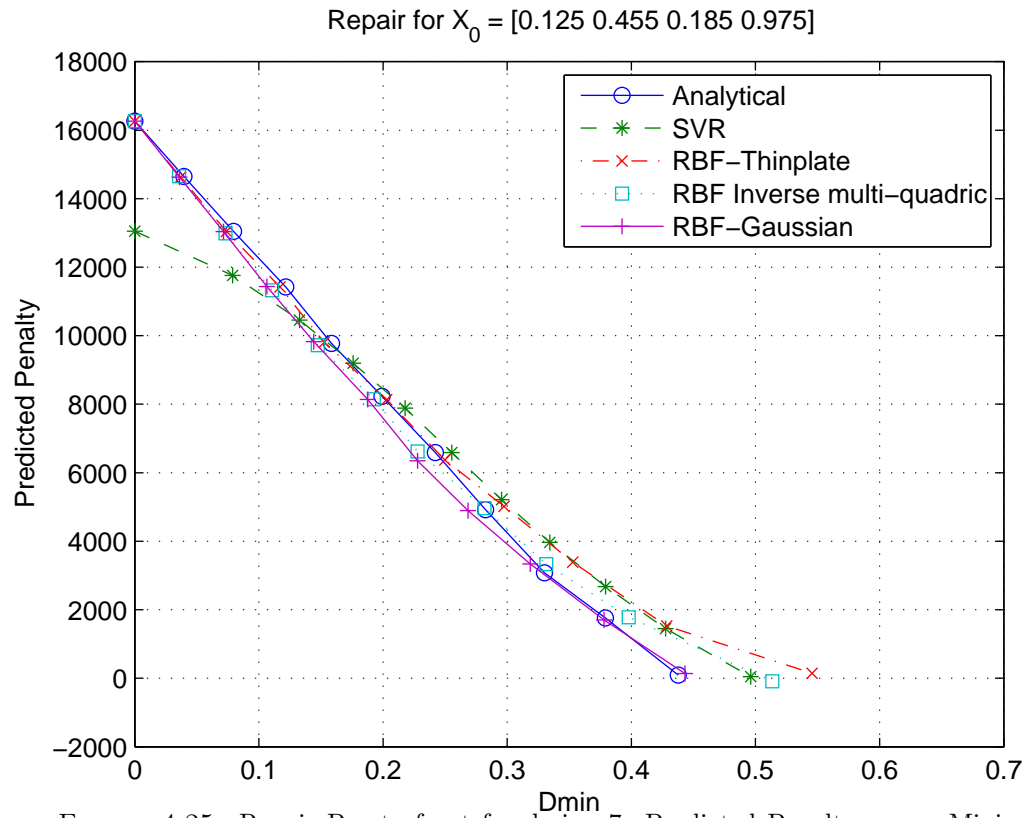


FIGURE 4.25: Repair Pareto front for design 7: Predicted Penalty versus Minimum distance

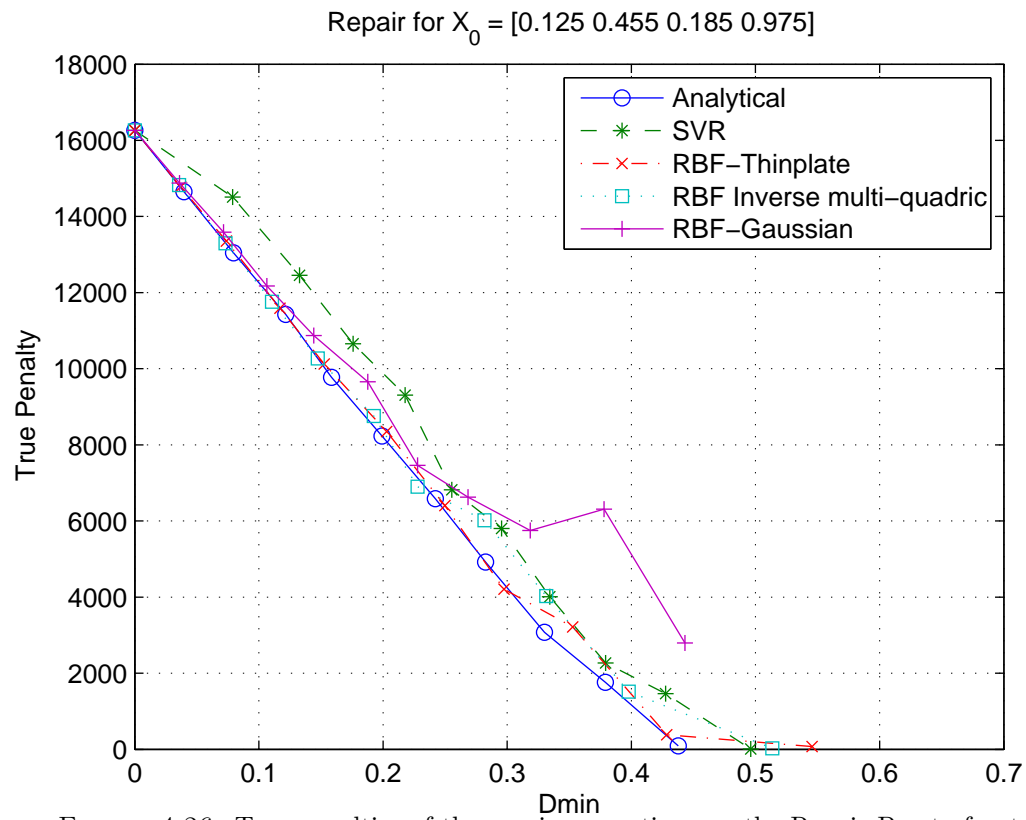


FIGURE 4.26: True penalties of the repair suggestions on the Repair Pareto front for design 7

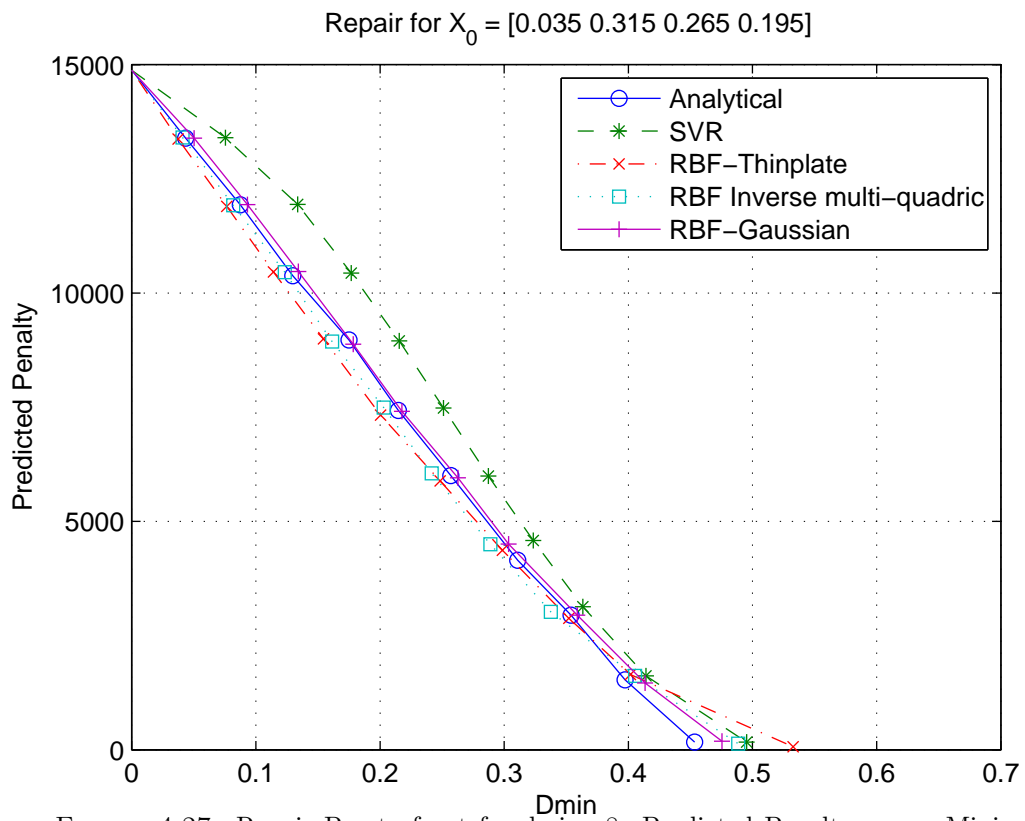


FIGURE 4.27: Repair Pareto front for design 8: Predicted Penalty versus Minimum distance

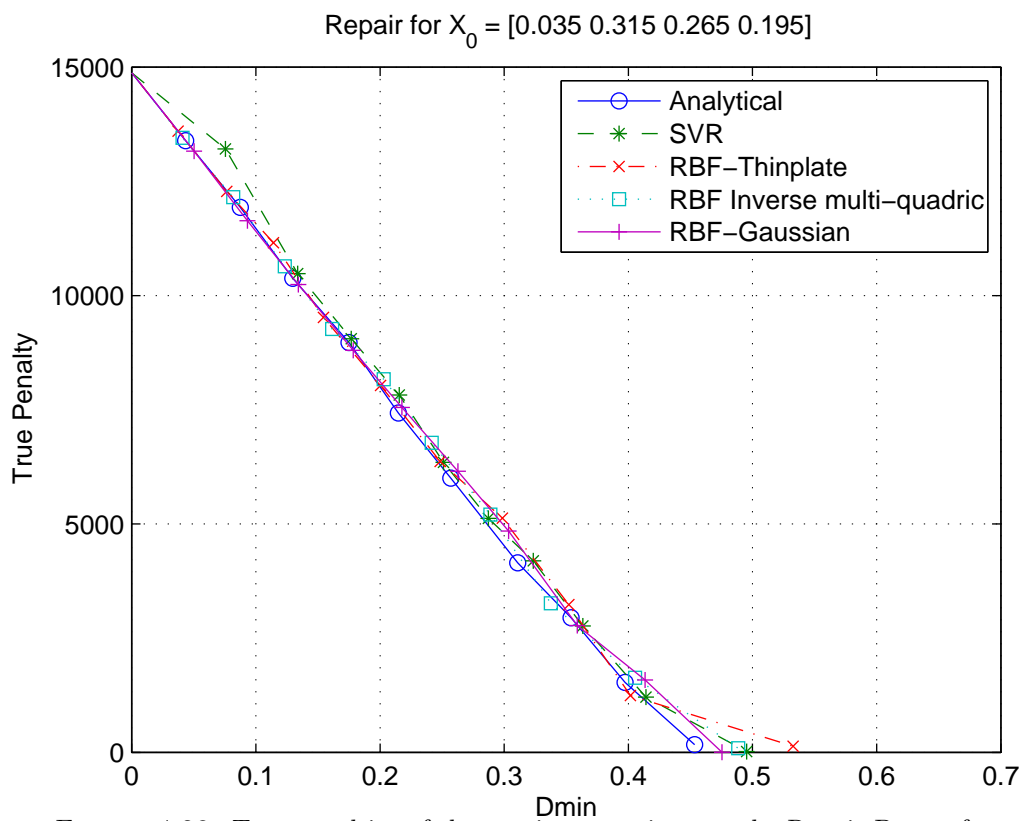


FIGURE 4.28: True penalties of the repair suggestions on the Repair Pareto front for design 8

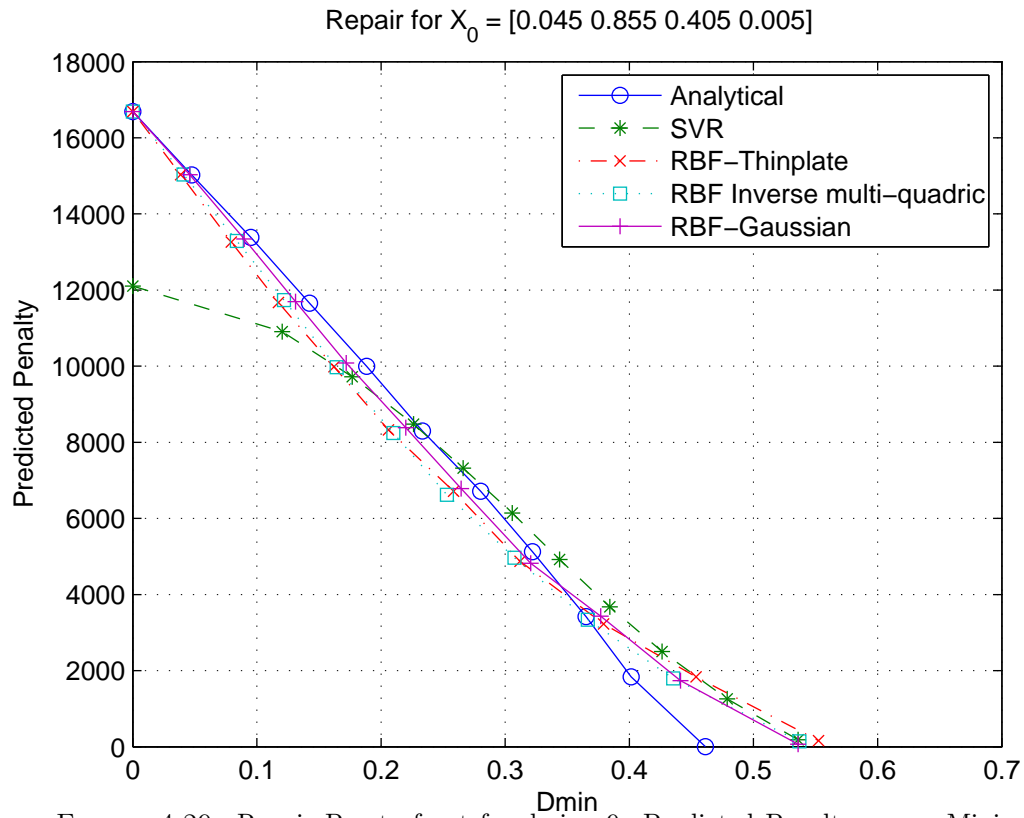


FIGURE 4.29: Repair Pareto front for design 9: Predicted Penalty versus Minimum distance

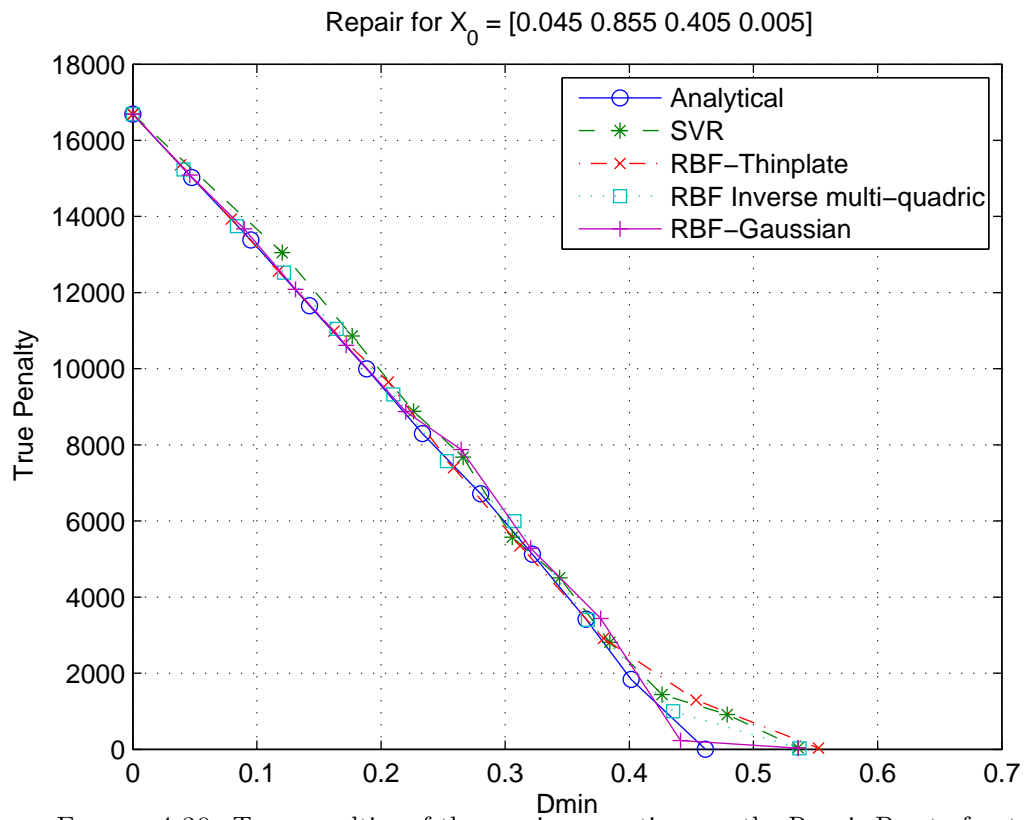


FIGURE 4.30: True penalties of the repair suggestions on the Repair Pareto front for design 9

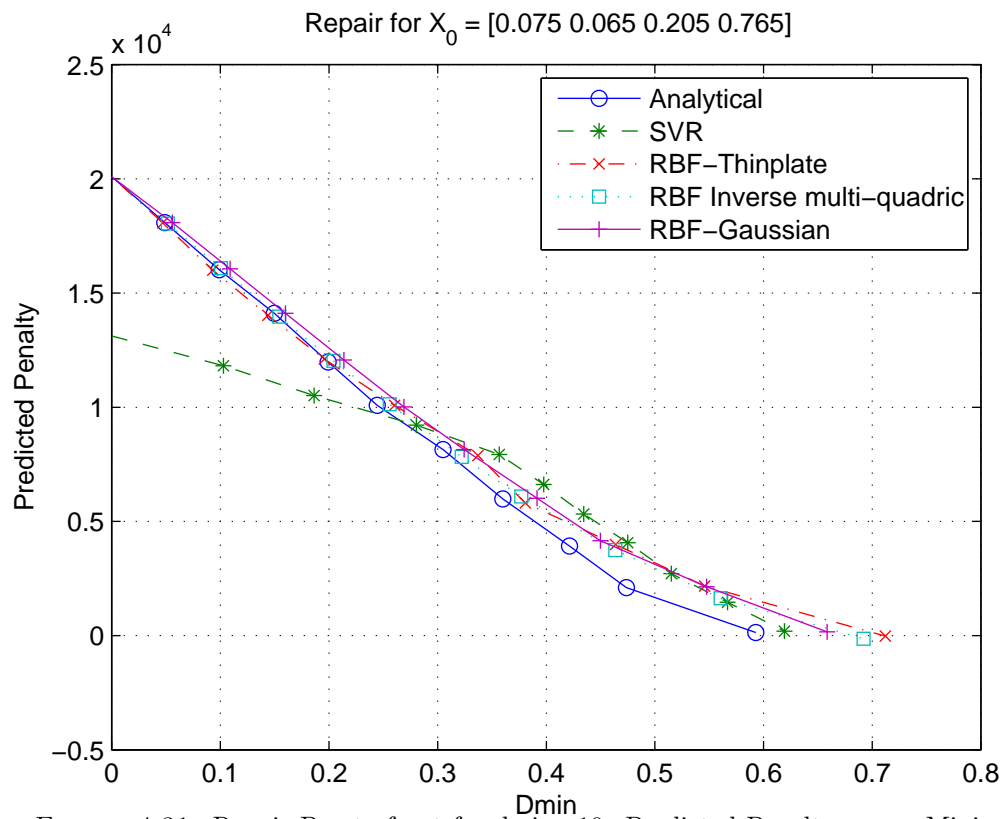


FIGURE 4.31: Repair Pareto front for design 10: Predicted Penalty versus Minimum distance

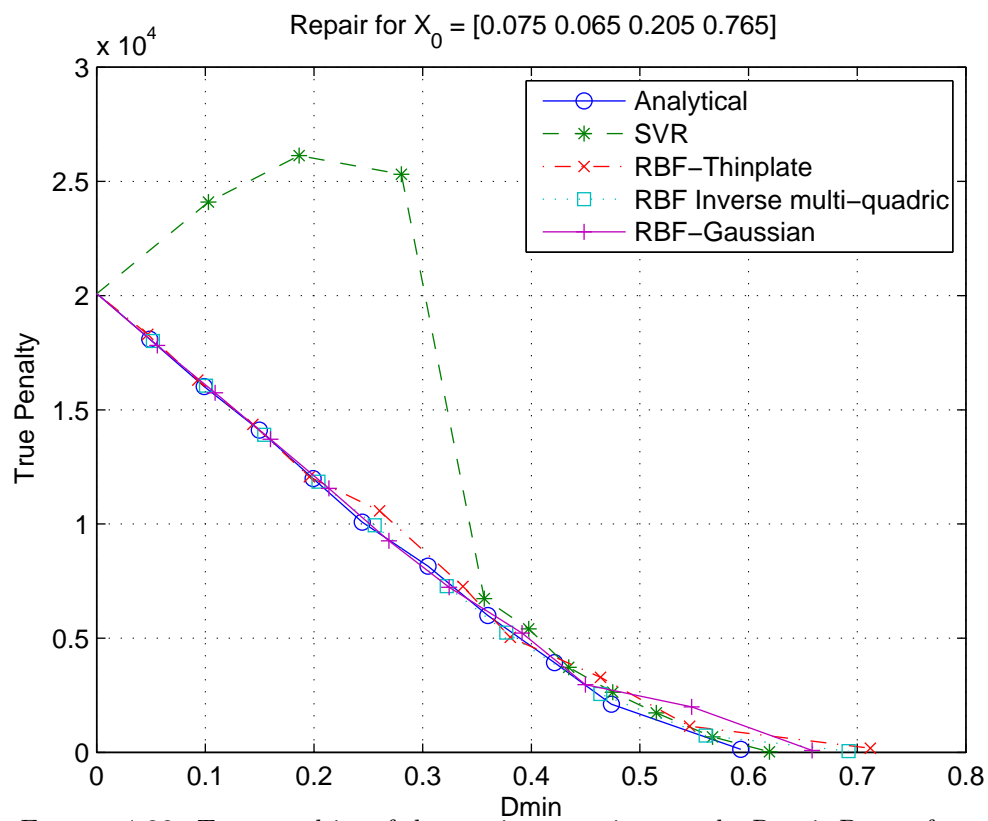


FIGURE 4.32: True penalties of the repair suggestions on the Repair Pareto front for design 10

The quality indicators  $Q$ , the average quality indicator  $\overline{Q}$  ( $\overline{Q} = Q/n$ ) and the standard deviation  $\sigma$  of the penalty differences for each of the four surrogate models are presented in Table 4.4. In all the cases,  $n = 110$  because there are 11 available data points (ten repair alternatives and 1 original design) on each of the 10 graphs.

TABLE 4.4: Quality comparison of the surrogates

Surrogate	$Q$	$n$	$\overline{Q}$	$\sigma$
SVR	$1.6623 \times 10^5$	110	$1.5112 \times 10^3$	$2.8634 \times 10^3$
RBF (thin plate)	$4.6445 \times 10^4$	110	422.2229	369.0119
RBF (Inverse multi-quadric)	$3.8468 \times 10^4$	110	349.711	345.7447
RBF (Gaussian)	$5.5695 \times 10^4$	110	506.3137	630.8099

It can be seen from Table 4.4 that the differences between different surrogates are limited. Repair alternatives generated by the RBF surrogate with inverse multi-quadric basis are most similar to the repair alternatives suggested by the original penalty and are most consistent. The repair alternatives given by the SVR surrogate are the least alike to the repair alternatives suggested by the original penalty and the least consistent.

## 4.6 A graphical user interface

A Graphical User interface (GUI) has been developed for the real-time display of the repair result and an interaction interface with the chief engineer who makes the decision in MATLAB.

Figure 4.33 shows an example of the GUI being used. On the left upper corner is a display window of the original design. The corresponding design variable set is displayed below the window. Manual input can be accepted to make immediate changes to the original design. On the left lower corner is a panel for repair setup. The engineer can specify the number of repair alternatives he wishes to view in the text box. By pressing the "Decide penalty ranges for me" button, the GUI can automatically decide a optimised penalty range for the user. The feasibility threshold that each repair alternative uses will spread uniformly in the specified penalty range, whose minimum is set to zero, and maximum is set to the predicted penalty value of the original design. The penalty range can also be manually modified by typing values into the two text boxes at the both end of the slider. The "Generate" push button will call the repair callback function in the background and find out the optimised repair alternatives before storing the repair information in a file. The search and optimisation process may last for several minutes, depending on the size of the optimisation problem. After the search is finished, the first repaired design is displayed immediately on the main window in the middle of the GUI. The slider can be dragged to change the penalty value threshold. The repair alternative

with the closest penalty value will be displayed in the window. Correspondingly, the repair design variable set will change in a real-time fashion.

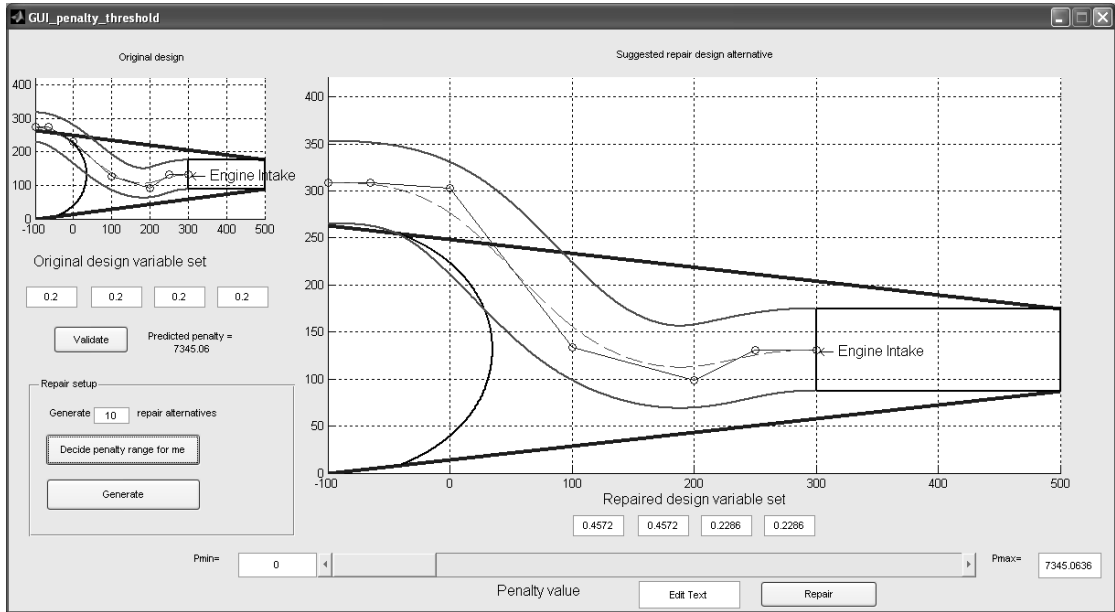


FIGURE 4.33: A MATLAB GUI for generating and examining repair alternatives

This visualisation kit enables the engineer to observe the repair alternative suggestions directly. The user can easily specify the penalty range to observe a series of repair alternatives. By developing this tool, it is hoped that the end user can make quick and more informed choice between robust and flexible design options even without knowing the mathematical and programming details in the background.

## 4.7 Summary

In this chapter, an automatic 2D engine intake duct model design workflow is set up. The modelling of knowledge is demonstrated by formulating three pieces of physics and engineering based design tips into penalty functions. After the repair system is fed into with the training data, it exhibits clear ability to repair faulty designs. The repair suggestion moves gradually from the original design, as the penalty threshold decreases. A comparison of the penalty prediction landscapes reveals the similarity between the resulting surrogate generated using different statistical models, and successful repair can be achieved using these surrogates.





## Chapter 5

# Aero-engine Intake Design Case with Aerodynamic Properties Calculated and Incorporated in the Knowledge Base

The aerodynamic properties of the engine intake are of great interest in the design process. Flow separation and distortion under cross wind or tail wind conditions need to be minimised, because they could cause undesirable vibration of rotating blade rows or engine surge. These aerodynamic properties could be predicted by a CFD application. If CFD results could be obtained and used to enhance the existing knowledge base, the knowledge base would be able to tell the quality of the design candidate from an aerodynamicist's view, and potentially repair or eliminate aerodynamically invalid designs. The chapter demonstrates how to obtain useful CFD results and integrate them into an existing knowledge base. The engine intake model used in Chapter 4 is reused in this chapter. The pressure distribution at the intake outlet is predicted using FLUENT, a common flow modelling simulation software.

### 5.1 The incorporation of flow modelling and simulation

In this section, the process for setting up the flow modelling and simulation is described. The process starts with exporting the model from MATLAB to the meshing software GAMBIT. The meshing file generated by GAMBIT is then exported to the CFD software FLUENT. Finally, the FLUENT result should be properly translated and incorporated into the existing knowledge base.

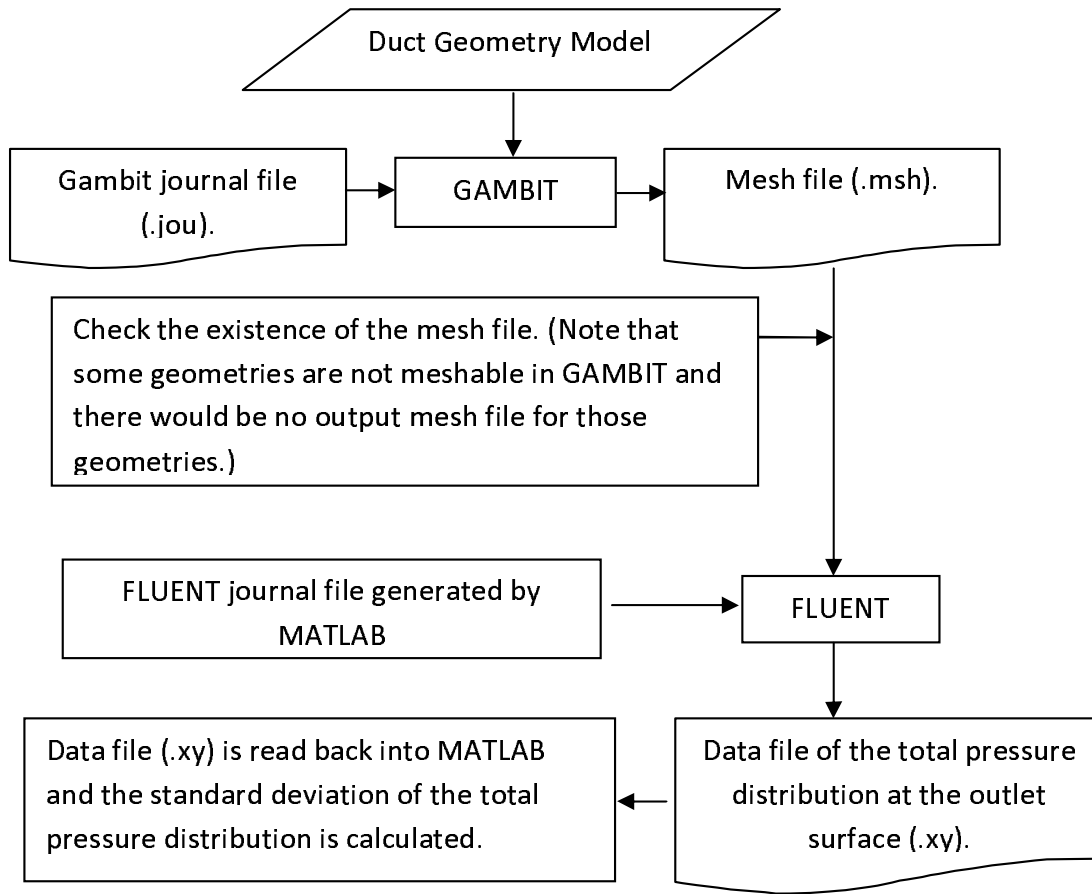


FIGURE 5.1: Determination of aerodynamic properties

### 5.1.1 Meshing process and the corresponding GAMBIT journal file

After a duct model is generated in MATLAB, two matrices, one representing the lower duct wall and the other representing the upper duct wall, are stored in the MATLAB workspace. From each of the two matrixes 21 points, including the starting and finishing points of the matrices, are sampled at a uniform interval. The resulting 42 points are created in GAMBIT as vertexes. The first 21 points which represent the lower duct wall are linked consecutively as an edge in GAMBIT. The same is done to the remaining 21 points, which represent the upper duct wall. The two starting points of the wall are linked as an edge and physically represent the flow inlet. The two finishing points of the wall are linked and physically represent the outlet. After the geometry is fully defined in GAMBIT, a boundary layer mesh of 6 rows is generated for both the lower and upper duct wall and a quadrilateral mesh is generated for the interior of the geometry. The boundary layer first-row height<sup>1</sup> is set to 1, and the growth factor<sup>2</sup> is set to 1.2. The

<sup>1</sup>The first-row height specifies the distance between the edge or face to which the boundary layer is attached and the first full row of mesh nodes.

<sup>2</sup>The growth factor represents the ratio  $b/a$  where  $b$  is the distance between the first and second full rows and  $a$  is the height of the first row.

interval of the face mesh is set to 5. Next, the physical nature of each of the components of the geometry are defined, i.e. the lower and upper duct wall are defined as “walls”; the edge that represents the inlet and outlet are defined as “velocity inlet” and “outlet”; the inside of the duct is defined as “fluid”. Finally, the work is exported to a two-dimensional mesh file (with an extension of .msh) which is readable by FLUENT.

A journal file which can carry out the process as described above and can be read into GAMBIT is generated by the codes as shown in Appendix A.5 and a sample journal file is listed in Appendix A.6. In this sample journal file, the name 147.msh is used as this mesh is generated based on the 147<sup>th</sup> geometry out of the 200 geometries which were used to construct the geometry quality knowledge base. Other mesh files were prepared in the same manner and named using the same naming rule. Figure 5.2 illustrates a successful meshing, which is based on the 147<sup>th</sup> geometry. The 5 layers of boundary layer cells and the face mesh are both discernable in the figure. Note that the geometries that are unphysical, for example, those that exhibit loops in the lower or upper walls, would not be meshed and there would not be any mesh file for those unphysical geometries, e.g. Figure 5.3, which is based on the 76<sup>th</sup> geometry. Therefore, for those unphysical geometries, it would not be possible to predict their flow behaviours or find their fan face distortion, and the process terminates here. Of these 200 geometries, 83 failed to be meshed.

### 5.1.2 CFD simulation process and the corresponding FLUENT journal file

If the geometry in question is successfully meshed, the corresponding mesh file can be read into FLUENT and a flow model can be set up and solved. After the solution is obtained, the total pressure distribution at the outlet of the duct can be saved as a data file. The model in question is firstly pre-processed by being scaled to one thousandth because the FLUENT default unit is in metre and the model is defined in millimetre. At a typical cruise height of 35,000 feet, the atmospheric pressure is 23.8 kPa and air temperature is 288.15 K [NASA (2010)]. In FLUENT the inlet is set as a velocity inlet, with a typical cruise speed of 200 m/s. The air viscosity is set to abide by the Sutherland viscosity law [Sutherland (1893)]. Then a  $\kappa - \epsilon$  turbulence model is defined in order to take viscous effect into consideration. The residual value of the continuity, velocity and energy, which are used as criterions of convergence are each modified to the smaller values of  $10^{-6}$ ,  $10^{-3}$  and  $10^{-3}$  respectively, in order to obtain more accurate results before convergence criterions are reached. The gauge pressure of the inlet is set to 1.5 times atmospheric pressure at 151987 Pascal. The turbulence is specified by intensity and hydraulic diameter, which are set to 5% and 0.0875 metre each. Note that 0.0875 metre is actually the diameter of the duct inlet and outlet surface because in this round duct, the hydraulic diameter equals the diameter of the duct. The model is then

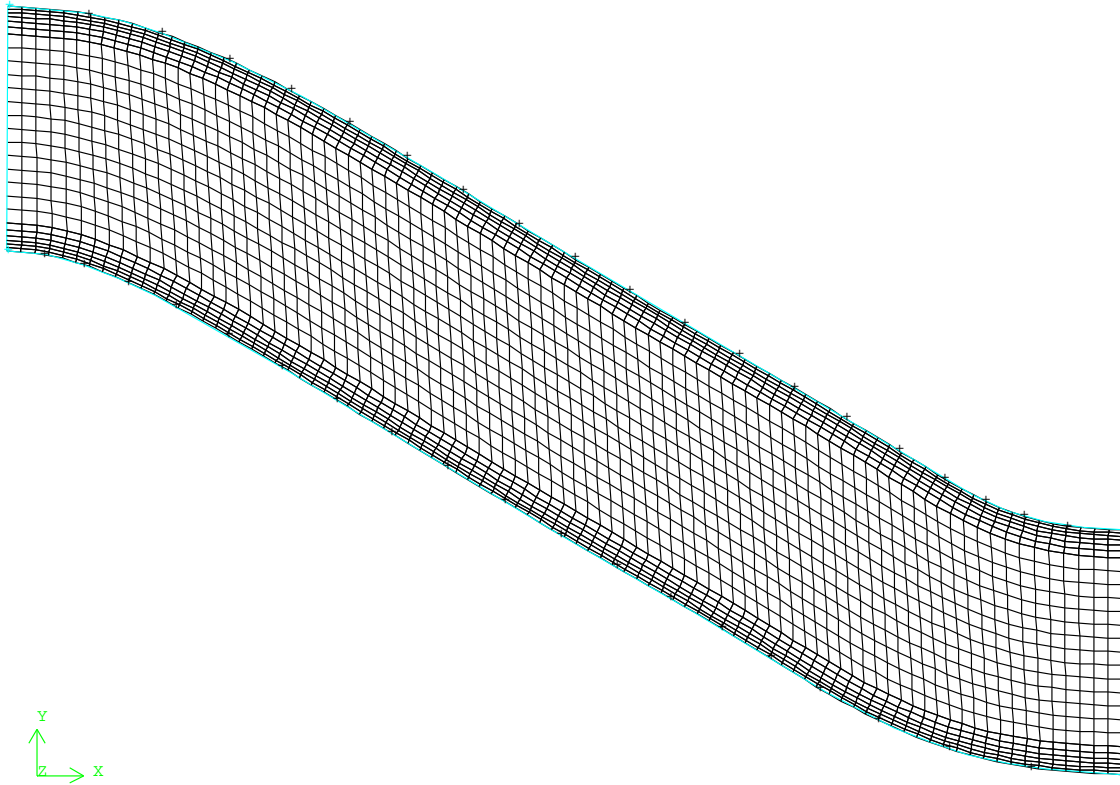


FIGURE 5.2: An example of successful meshing

initialised and iterated 200 times. The pre-process proved to be effective for solving most of the models with 112 out of 117 meshes successfully solved in the test. After the flow is predicted, the values of the total pressure at each node of the *outlet* face are recorded. The data will be read back into MATLAB and the standard deviation of the pressure will be calculated and used as a metric of the fan face distortion.

FLUENT has a scripting facility which gives the opportunity to accomplish the necessary operations as described above in an automatic fashion by using a journal file. To automate the design cycle, the FLUENT journal file is in turn automatically generated by a piece of MATLAB code as shown in Appendix A.5. A sample of the resulting FLUENT journal files is shown in Appendix A.7.

The flow prediction of the total pressure distribution (for model No.9) is shown in Figure 5.4.

The total pressure distribution at the outlet of model No.9 is shown in Figure 5.5.

The residual monitor in FLUENT shows the residual convergence history of the case (Figure 5.6). The case of Geometry 9 is converged after 568 iterations.

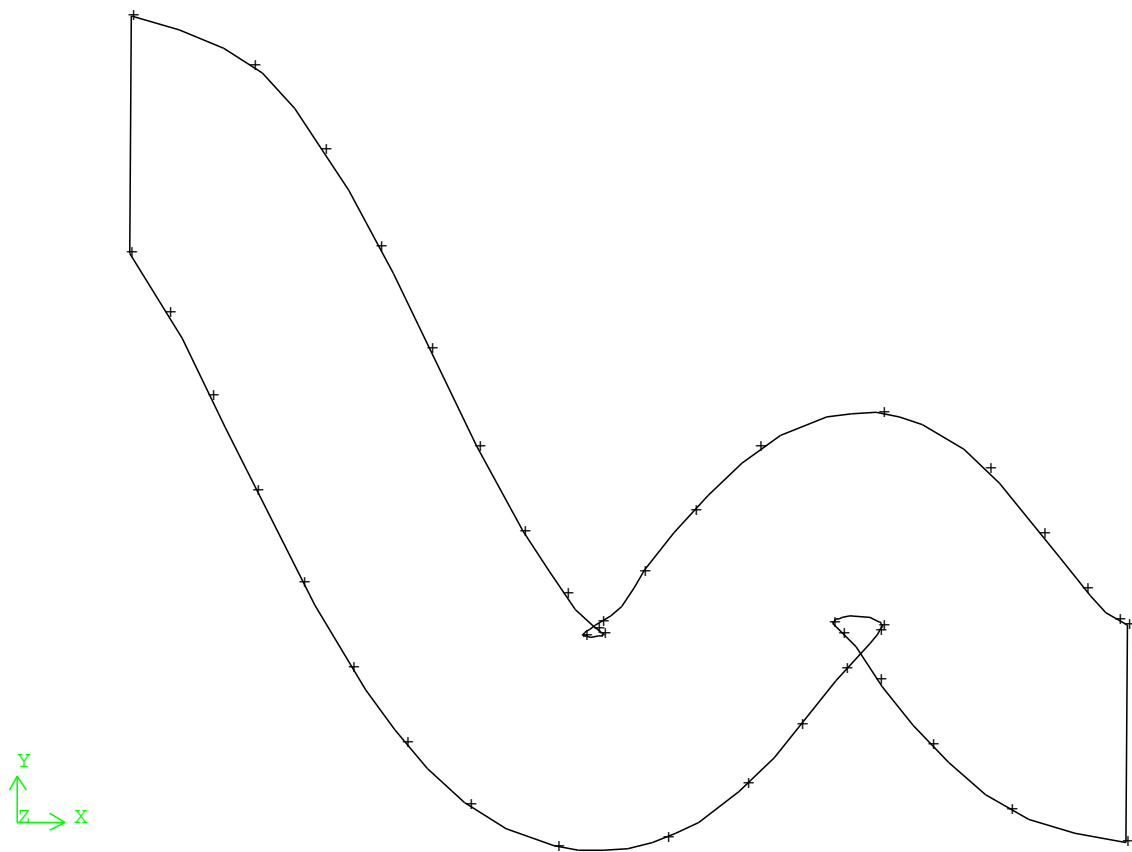
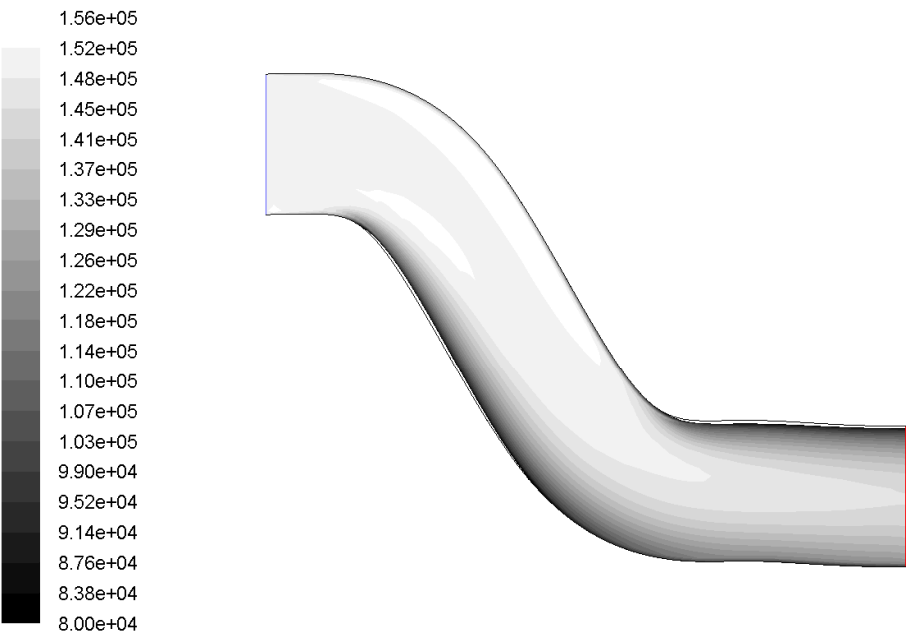


FIGURE 5.3: An example of meshing failure



Contours of Total Pressure (pascal) Jul 02, 2010  
FLUENT 6.3 (2d, pbns, ske)

FIGURE 5.4: A typical distribution of the total pressure in the intake

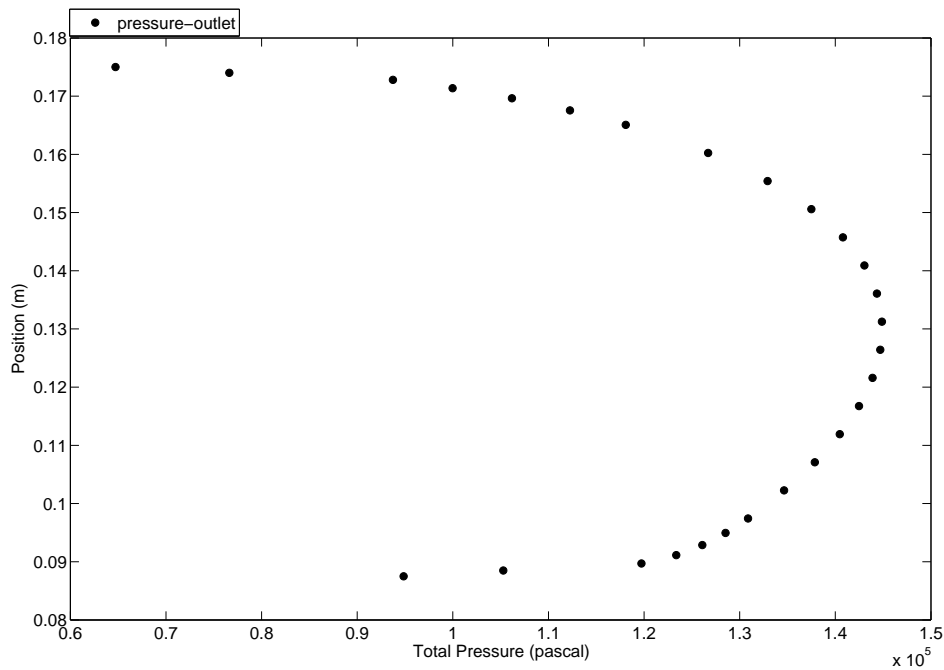


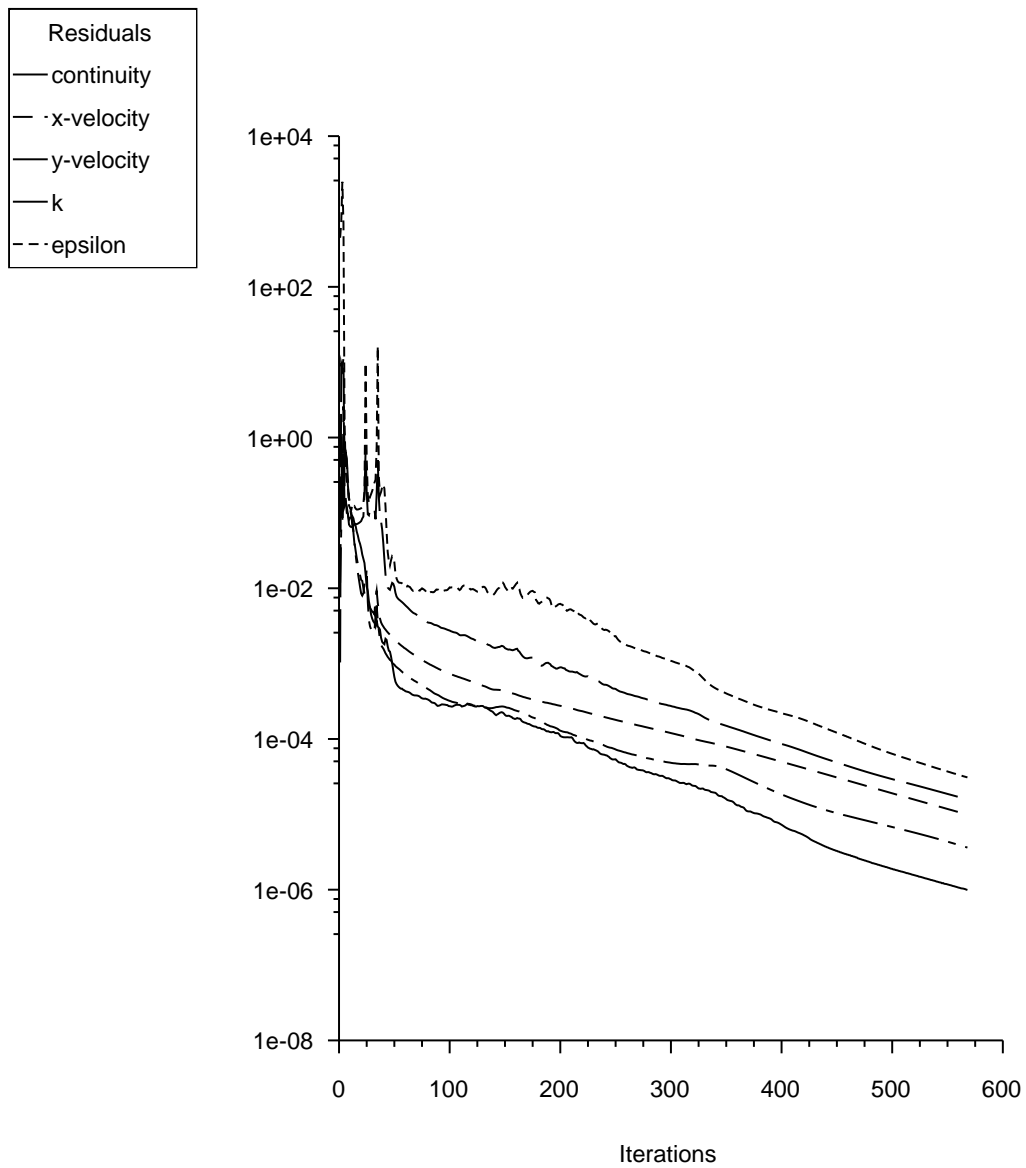
FIGURE 5.5: A typical distribution of the total pressure at the engine face

The plots of velocity streamlines (Figure 5.7) are calculated in FLUENT to understand the flow pattern.

Figure 5.8 illustrates the range of total pressure distribution at the outlet of the intake ducts which are in the sampling plan established in Section 5.3. Only the total pressure distribution profiles of those designs which can be solved in FLUENT are presented. The pressure profiles corresponding to that of Geometry 8 and Geometry 9 are both highlighted in the figure.

## 5.2 Verification of the CFD process

Analysis result could change when the fineness of the mesh is altered. Generally the finer the mesh, the smaller the simulation error is, but the longer the computation time it takes. When an analysis activity involves many CFD simulations (e.g., a sampling plan evaluation), one may wish to use a coarse grid to speed up the process. However the mesh should not be so coarse that the effectiveness of the CFD simulation is jeopardised. In this work the purpose of the CFD analysis is to find a measure for the pressure distortion at the intake outlet. As discussed in Section 5.1, A FLUENT process has been set up and the total pressure value at 21 evenly distributed points at the outlet face has been collected. Then the standard deviation of the 21 values is used as a measure for the distribution. The measure is then normalised and incorporated in the penalty table as

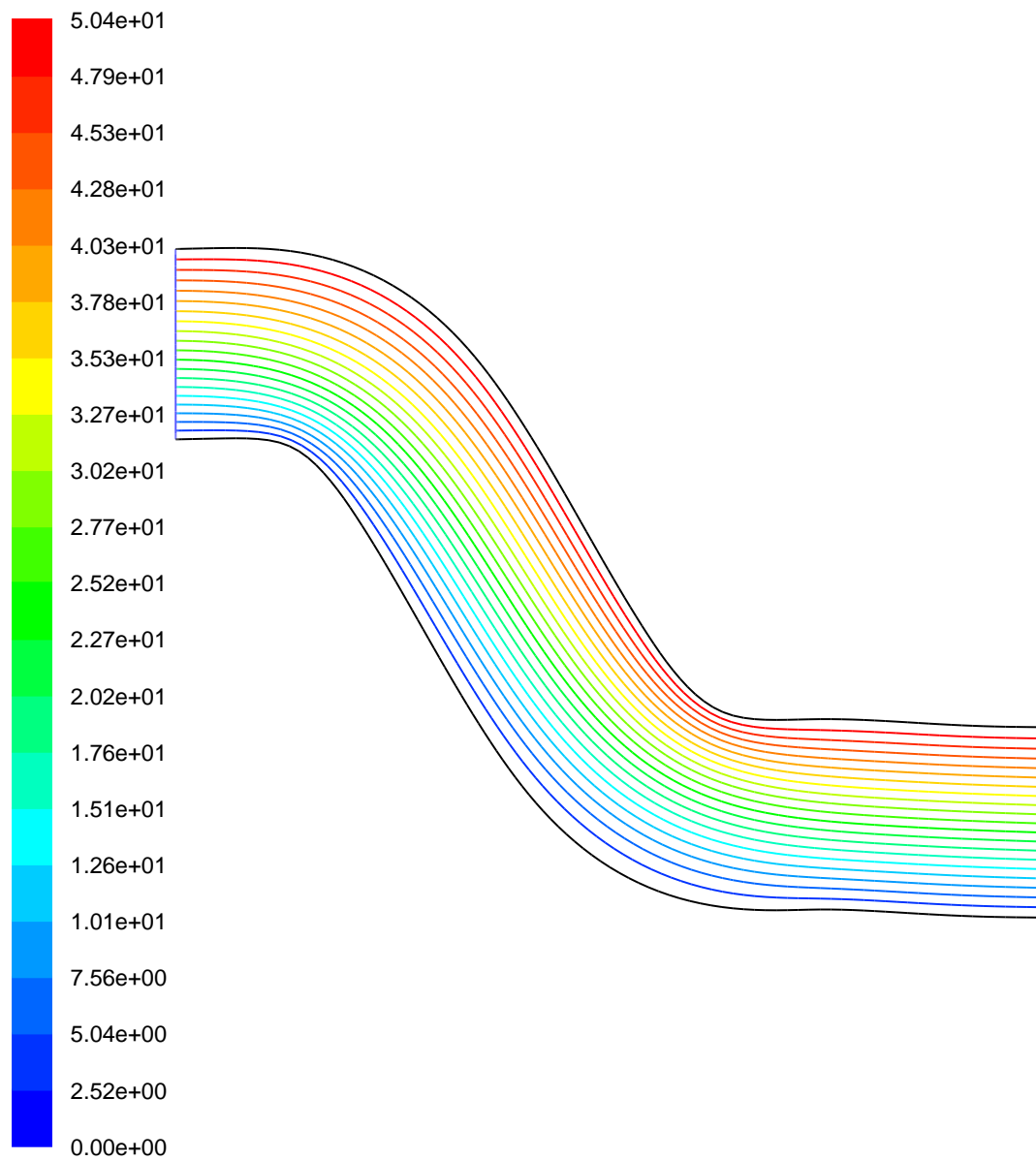


Scaled Residuals

Apr 28, 2011  
FLUENT 6.3 (2d, pbns, ske)

FIGURE 5.6: Residual convergence history





Contours of Stream Function (kg/s)

Apr 28, 2011

FLUENT 6.3 (2d, pbns, ske)

FIGURE 5.7: Velocity streamlines

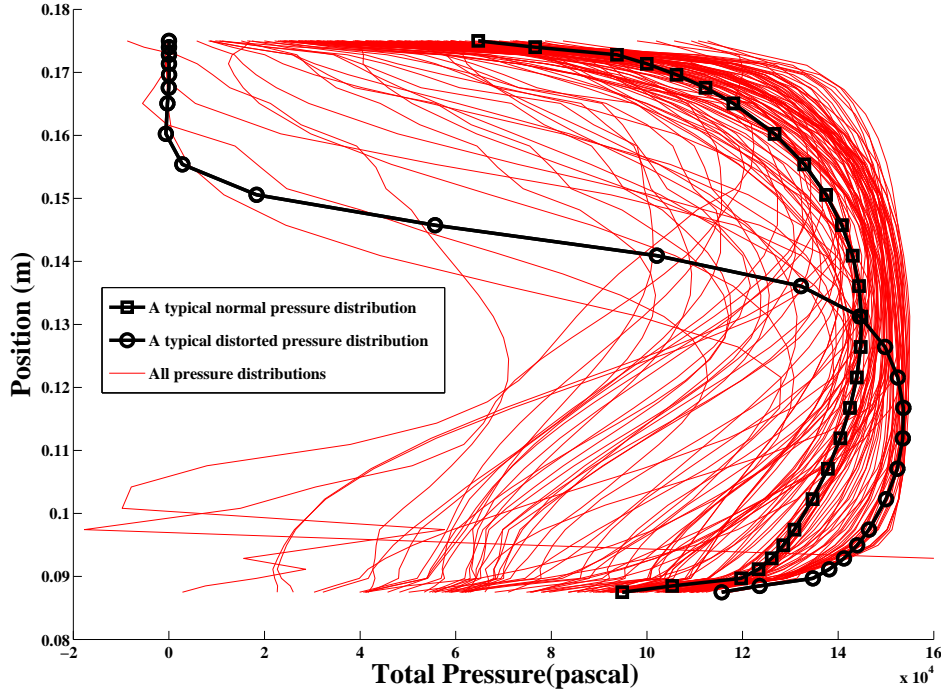


FIGURE 5.8: Total pressure distribution profiles at the outlet of the intake ducts

a measure of the aerodynamic performance of the design. Therefore, what is required in this work is a *relative* measure which is effective in telling the difference between the distortions of two designs, rather than the accurate value of the standard deviation of the pressure data, or the accurate value at each of the discrete points at the outlet face. To verify that the grid scheme that is used is effective in telling the difference of the distortion effect, alternative meshing schemes have been set up, results being compared against the ones obtained with the original meshing setup in Section 5.1.

### 5.2.1 Effect of using smaller $y^+$ value

Due to the no-slip conditions, there is a boundary layer at the near-wall region where solution variables have large gradients compared with the core, turbulent flow [Day (1990)]. The  $k-\varepsilon$  model used in this work is primarily valid for turbulent core flows. Therefore semi-empirical formulas called “wall functions” are provided by FLUENT and are used in this work to more accurately represent the flow in this boundary region. It is recommended that for standard wall functions, each wall-adjacent cell’s centroid satisfy  $30 < y^+ < 300$ , where  $y^+$  is called the wall unit, and is defined by

$$y^+ \equiv \rho u_r y / \mu \quad (5.1)$$

where  $\rho$  is the density of the fluid,  $y$  is the distance between the wall and wall-adjacent cell's centroid,  $u_r$  is the fluid velocity, and  $\mu$  is the viscosity of the fluid. The interested reader can refer to [FLUENT (2006a)] for more details. It was found by the author that such a  $y^+$  value requires a very small first-row height  $a$  (approximately on the order of magnitude  $\sim 2$ ) of the boundary layer to be specified in the meshing process, which would potentially increase the grid density and computational time. (Remember in Section 5.1.1,  $a = 1$  was used.) It is an interesting question whether the relative measure of outlet pressure distortion obtained from CFD analysis would change when using a smaller  $y^+$  value.

Two typical geometries were chosen from the 200 samples. Geometry 147 is roughly a straight duct (Figure 5.2) and Geometry 148 (Figure 5.9) is a reverse S-shape design with an additional twist at the front of the duct. Intuitively the more convoluted design Geometry 148 causes more pressure distortion than the straight one.

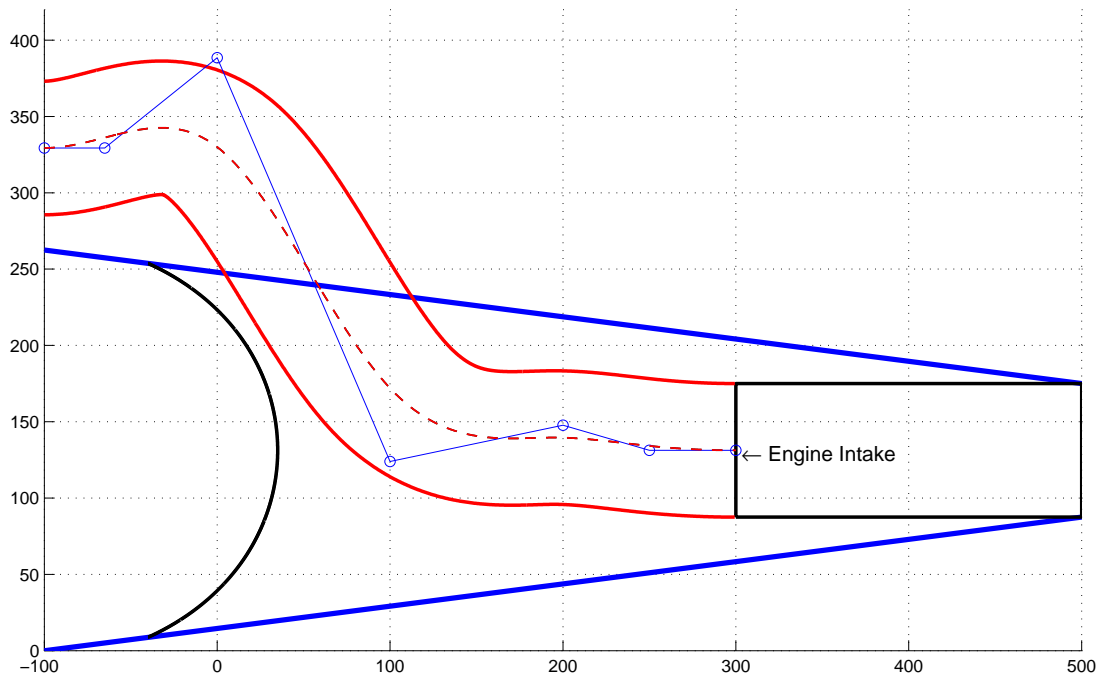
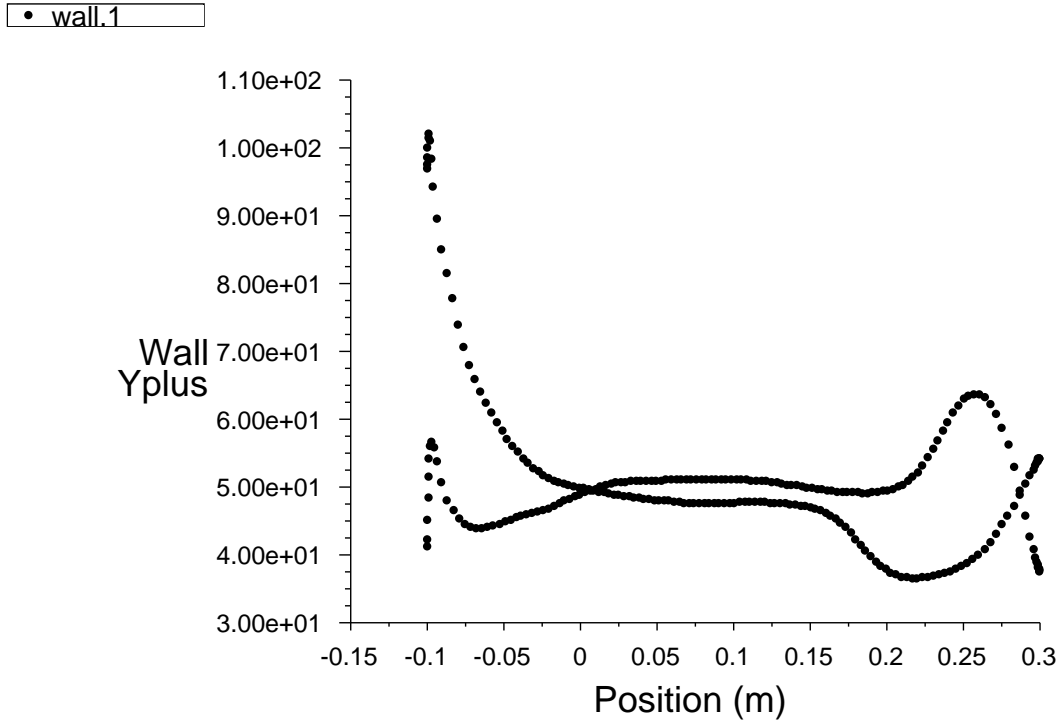


FIGURE 5.9: Geometry 148

To generate a small  $y^+$  value within the range of  $30 < y^+ < 300$  for Geometry 147,  $a = 0.08$  was used.  $y^+$  value varies along the wall between 36 and 100 (Figure 5.10).

The  $y^+$  value is geometry dependent. When using the same meshing setup  $a = 0.08$  for Geometry 148, a different distribution is obtained with the  $y^+$  values between 21 and 102 (Figure 5.11). According to [FLUENT (2006a)], the mesh should be made either coarse or fine enough to prevent the wall-adjacent cells from being placed in the buffer layer where  $5 < y^+ < 30$ .



Wall Yplus

Apr 13, 2011  
FLUENT 6.3 (2d, pbns, ske)

FIGURE 5.10:  $y^+$  distribution along the duct walls for Geometry 147

In order to find a proper  $a$  which would limit the  $y^+$  in the recommended range,  $a$  is manually set to 0.05, 0.07, 0.1, 0.15 and 0.2. Among these values, 0.1 was the most promising one, as it was expected that a slight increment in  $a$  would result in a slight, proportional increment in  $y^+$  when Equation 5.1 is examined. However it proved that none of these values could lead to a satisfactory  $y^+$  range along the wall cells.

$a = 0.08$  is used for both geometries as a expediency when the small  $y^+$  scheme is required. And the corresponding standard deviation value for each scheme is calculated and presented in Table 5.1

	Standard deviation of the outlet pressure	
	original meshing scheme	small $y^+$ meshing scheme
Geometry 147	$1.29 \times 10^4$	$3.97 \times 10^4$
Geometry 148	$2.28 \times 10^4$	$4.33 \times 10^4$

TABLE 5.1: Standard deviation obtained using different  $y^+$  meshing schemes

It could be seen that both the original meshing scheme and the small  $y^+$  meshing scheme indicate that Geometry 147 causes less pressure distortion at the fan face than Geometry 148. The distortion values increase by a factor of 3.07 for Geometry 147, and 1.89 for

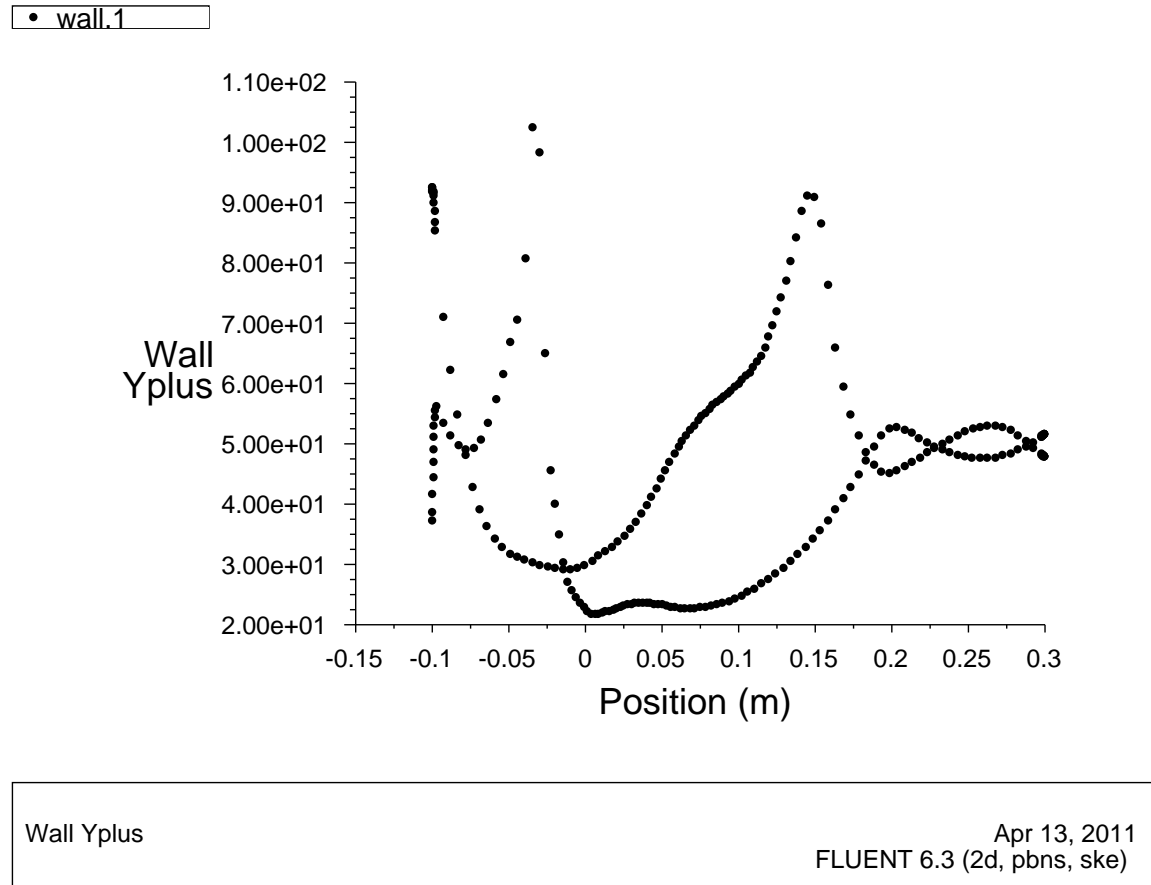


FIGURE 5.11:  $y^+$  distribution along the duct walls for Geometry 148

Geometry 148. Despite the increase, if the values are collected and normalised, the resulting relative measure of the aerodynamic performance of the design is similar for both schemes. To prove this, a  $y^+$  adaption process have been used in FLUENT to appropriately refine or coarsen the mesh along the wall during the solution process. Wall adjacent  $y^+$  value is specified in the range between 30 and 100. The resulting  $y^+$  values of four geometries were manually examined. The adaption was only effective for 3 of them.

The 200 geometries have gone through the simulation using  $y^+$  adaption, and a new set of 200 pressure data have been obtained. The data is processed by the same way as in Section 5.3. The penalty related to the pressure distortion is calculated and compared to that obtained from the previous simulations.

The new penalty data set is very similar to that obtained from the previous simulations (Figure 5.12), with goodness of fit  $R^2 = 0.8364$ . The difference in the penalty sum is even smaller, (Figure 5.13), with goodness of fit  $R^2 = 0.9681$ .

To conclude, using a small  $y^+$  hardly change the relative measure of pressure distortion. The meshing is made finer with a small  $y^+$ , which requires more computational time. It is also difficult in practice to maintain  $30 < y^+ < 300$  in an automatic process with

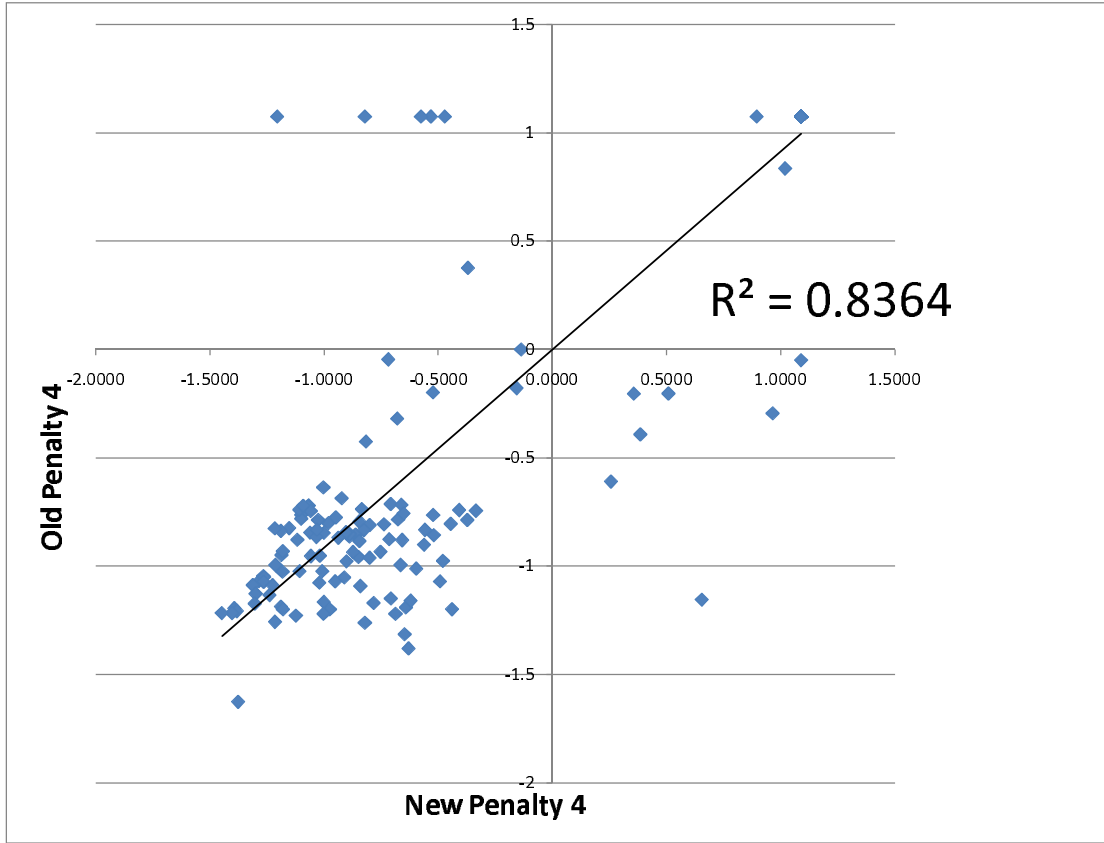


FIGURE 5.12: Goodness of fit plot of the Penalty 4

a fixed first row height  $a$ . For these three reasons, a less strict  $y^+$  is used in this work, as a sufficient method to get a relative measure of the pressure distortion. It should be noted the recommendation of  $30 < y^+ < 300$  should be used when an  $k-\varepsilon$  model is used with standard wall-functions *and* precise CFD results are required.

### 5.2.2 Grid convergence

As mentioned in the beginning of this section, coarser grids are desired when a large number of designs are to be evaluated. However a coarser meshing scheme causes more discretisation errors, which occur from the representation of the governing flow equations as algebraic expressions in a discrete domain of space [NASA (2008b)]. A mesh convergence study is typically used to determine the lowest feasible mesh resolution.

Geometry 147 was selected for the grid convergence study. Five refined meshing schemes were manually prepared for it. The total number of cells in each scheme was 2500, 5000, 10000, 20000 and 40000 respectively. The finer grid is made to double the previous coarser grid for the convenience of the following calculation of the grid convergence index (GCI). Each grid is assigned a number for narrative convenience (Table 5.2).

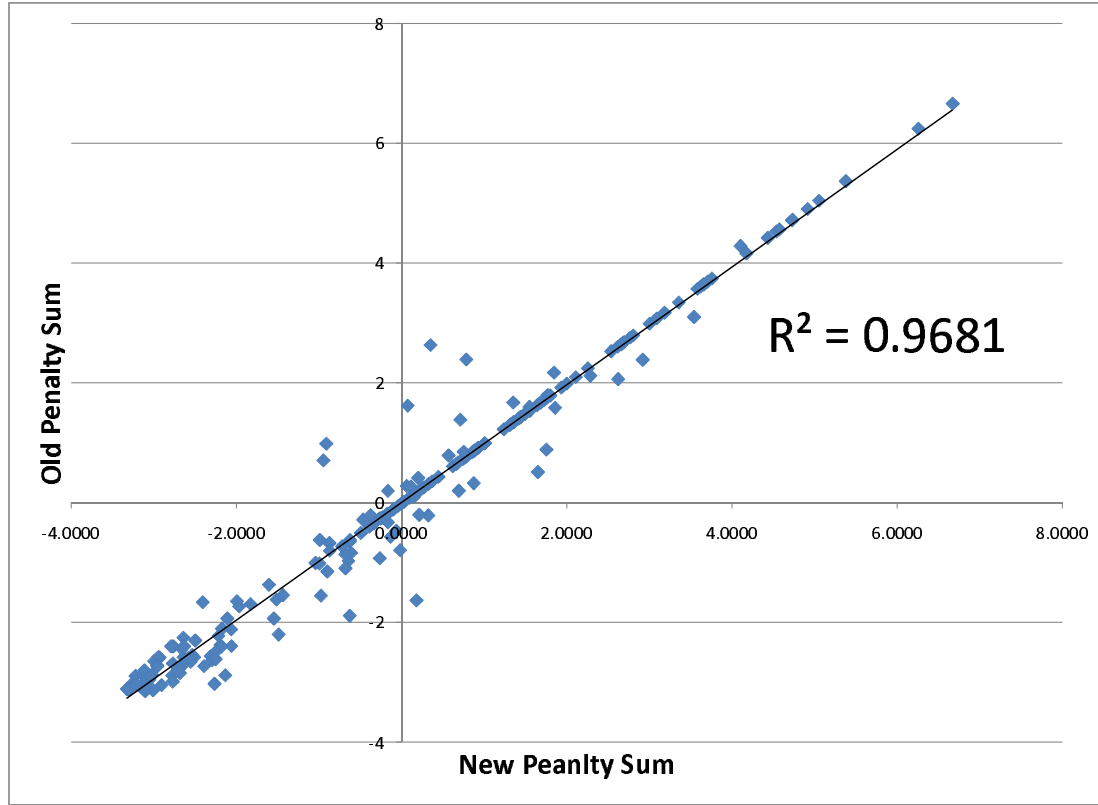


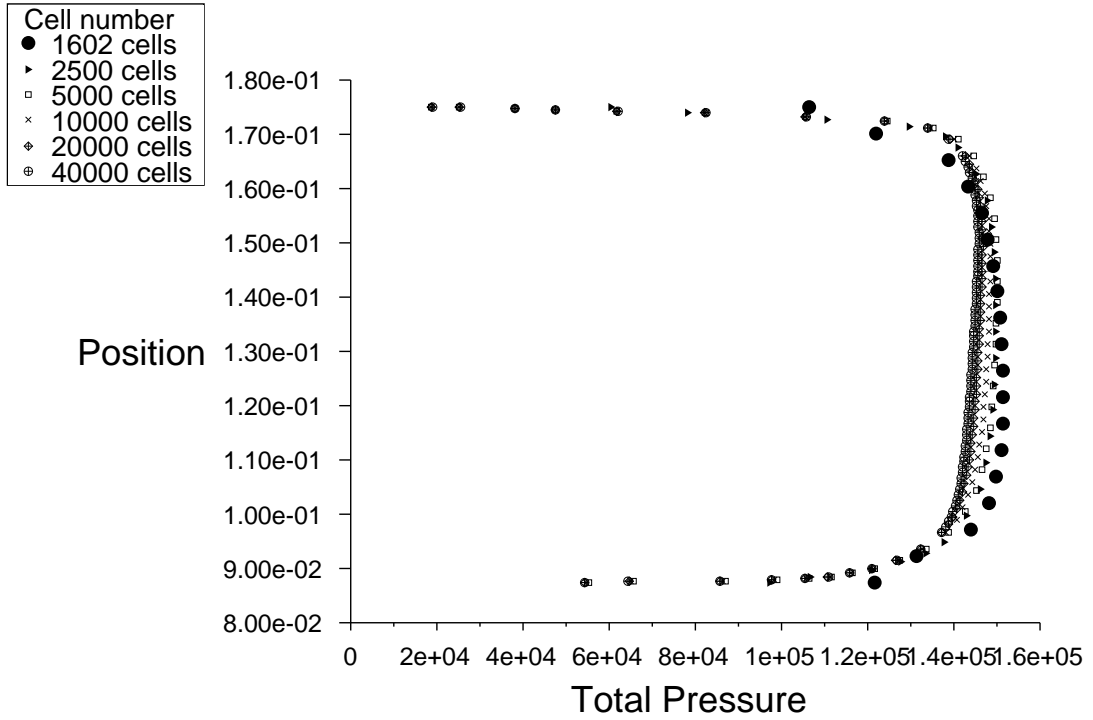
FIGURE 5.13: Goodness of fit plot of the penalty sum

The pressure distribution at the fan face for each meshing scheme has been presented in Figure 5.14. The mesh with 1602 cells used for the flow simulation in Section 5.1 is also listed for comparison. It is clear from Figure 5.14 that finer meshing schemes does not significantly change the prediction of the pressure distribution at the outlet. Numerically, the GCI could be calculated from the value of the simulation results to provide a consistent manner in reporting the results of grid convergence studies, and validate the simulations. The GCI is a measure of the percentage the computed value is away from the value of the asymptotic numerical value. A clear introduction and procedure could be found in [NASA (2008a)], and [Power et al. (2003)] is an example of a comprehensive set of convergence test. It is recommended in [NASA (2008a)] that the GCI computed using three levels of grid in order to accurately estimate the order of convergence.

The GCI on the fine grid is defined as

$$\text{GCI}_{fine} = \frac{F_s |\varepsilon|}{r^p - 1} \quad (5.2)$$

where  $F_s$  is a factor of safety and  $r$  is the grid refinement ratio. For comparisons over three or more grids,  $F_s = 1.25$  is recommended.  $r$  is defined by  $r = h_2/h_1$ , where  $h_1$  and  $h_2$  are spacings of grid with  $h_1$  being the finer (smaller) spacing. For a two dimensional



Total Pressure

Apr 14, 2011  
FLUENT 6.3 (2d, pbns, ske)

FIGURE 5.14: Comparison of pressure distribution with different grid scheme

problem where each consecutive mesh is twice as fine as the last, it could be estimated that  $r = \sqrt{2/1} = \sqrt{2}$ . This is known as the effective grid refinement ratio.

The relative error  $\varepsilon$  in Equation 5.2 is defined as

$$\varepsilon = \frac{f_2 - f_1}{f_1} \quad (5.3)$$

where  $f_i$  ( $i = 1, 2, 3$ ) are the solution functionals, such as the standard deviation of the fan face pressure distribution, or a distortion factor.

The order of convergence  $p$  in Equation 5.2 is defined as

$$p = \ln \left( \frac{f_3 - f_2}{f_2 - f_1} \right) / \ln(r) \quad (5.4)$$

For each grid, three functionals are calculated based on the total pressure data  $p_i, i = 1, 2, \dots, n$  at the fan face. These are the standard deviation of the pressure  $\sigma_p$ , average pressure  $\mu_p$ , and pressure distortion factor (DTMM), which is defined in [Hughes et al.



(1985)] as

$$DTMM = \frac{p_{\max} - p_{\min}}{\mu}. \quad (5.5)$$

	1	2	3	4	5	6
Cell Number	1602	2500	5000	10000	20000	40000
$\sigma_p$	1.29E+04	2.39E+04	4.00E+04	3.66E+04	3.27E+04	2.87E+04
$\mu_p$	1.42E+05	1.33E+05	1.18E+05	1.24E+05	1.28E+05	1.32E+05
DTMM	0.32	0.67	1.11	1.04	1.00	0.96

TABLE 5.2: Variation of simulation functionals on multiple grid levels

Grids 3, 4 and 5 are used for calculating GCI. Following Equations 5.2, 5.3 and 5.4, GCI and asymptotic rate for  $\sigma_p$ ,  $\mu_p$  and DTMM are calculated based on the data in Table 5.2, and presented in Table 5.3.

	GCI <sub>45</sub>	GCI <sub>34</sub>	Asymptotic rate
$\sigma_p$	-57.09%	-43.82%	1.0396
$\mu_p$	3.30%	5.04%	0.6978
DTMM	5.84%	7.89%	0.6758

TABLE 5.3: GCI and asymptotic rate for simulation functionals

It can be seen from Table 5.3 that the asymptotic rate for  $\sigma_p$  is approximately one and indicates that the solutions are well within the asymptotic range of convergence. Other solution functionals exhibit small errors and trend of convergence as well. Here,  $\sigma_p$ , the standard deviation of the fan face pressure is used as the indicator of the pressure distortion and later incorporated in the knowledge base.

$$P_4 = \sigma_p \quad (5.6)$$

### 5.2.3 Turbulence models

It is well-known that no single turbulence model is universally accepted as being superior for all classes of problems. In this work, a fully turbulent duct flow is investigated and therefore a standard  $\kappa - \epsilon$  turbulence model which is valid for fully turbulent flows has been used. Standard  $\kappa - \epsilon$  model is a two-equation model in which the turbulent velocity and length scales are independently determined by solving two separate transport equations. The model provides robust and reasonably accurate solution for a wide range of turbulent flows since it was proposed by Launder and Spalding (1972), and is popular in industrial flow and heat transfer simulations. When accurate prediction is required, and/or the available computational resources and time is limited, other turbulent models could be considered. The difference of various models, and their suitability are beyond the scope of the thesis, as the sole purpose of the CFD analysis in this chapter is to

demonstrate that the knowledge from CFD simulations could be extracted and used for the setup of a knowledge base. Without benchmark data at hand, it is not possible to measure the quality of results obtained from different turbulent models. Therefore, simulations are only carried out with a standard  $\kappa - \epsilon$  turbulence model. However the interested readers can refer to [FLUENT (2006b)] for more information on choosing turbulence models.

### 5.3 Sampling plan, penalty table and surrogate modelling

After the three penalty equations based on the engineering knowledge and one penalty based on the CFD result of the pressure distortion have been set up (Equation 5.6), 200 designs generated in a sampling plan have been analysed using the CFD setup described above. The knowledge surrogate model was then fitted to the resulting responses.

#### 5.3.1 Sampling plan

A good sampling plan should cover the design space in a thorough and uniform fashion. Here the method illustrated in [Forrester et al. (2008)] was adopted and a four dimensional sampling plan of 200 points was generated. Part of the sampling plan is presented in Table 5.4.

TABLE 5.4: Sampling plan

Sample	Design variables			
	#1	#2	#3	#4
1	0.8075	0.9125	0.2575	0.0425
2	0.4925	0.2075	0.2375	0.8575
3	0.8675	0.4675	0.7775	0.7275
...	...	...	...	...
199	0.0575	0.3525	0.1225	0.6875
200	0.4775	0.9275	0.1475	0.2125

#### 5.3.2 The penalty table

For each of the samples, penalty values based on the penalty functions that have been introduced in Section 4.3 and 5.1 are calculated and assembled in a penalty value table as presented in Table 5.5.

Note that for the 4<sup>th</sup> column, which is the standard deviation of the total pressure distribution on the fan face, some values are not available because the corresponding designs are not meshable in GAMBIT or solvable in FLUENT. For the meshable geometries,

the largest value of standard deviation found was  $7.37 \times 10^4$ , which indicated a relatively severe fan face distortion. As it is even worse to be not meshable or solvable (which is a sign of unphysical or very faulty designs) than to have large standard deviation values (which indicates the design was not very engineering favourable but could still be considered), the author endeavours to penalise those samples which failed to be meshed or solved with a large value of  $7.37 \times 10^4$ , which equals the largest standard deviation value from those samples that can be solved. The result is presented in the last column of Table 5.5 as “Modified std.”.

$$\text{Modified std.} = \begin{cases} 73666.9 & \text{if std.}=\text{NaN} \\ \text{std.} & \end{cases} \quad (5.7)$$

The averages and standard deviations of each penalty are shown in the lower half of Table 5.5.

TABLE 5.5: Penalties and statistics

Penalties					
Sample	Penalty 1	Penalty 2	Penalty 3	Standard deviation (std.)	Modified std.
1	373.2	1541.6	0	NaN	73666.9
2	21.6	2665.9	31887.9	NaN	73666.9
3	467.4	0	0	33117.5	33117.5
4	24.5	54.6	0	57825.9	57825.9
5	3359.9	0	11195	22210.4	22210.4
...	...	...	...	...	...
196	96.5	387.4	0	31052.0	31052.0
197	116.2	0	321.1	44693.1	44693.1
198	628.6	0	0	22695.3	22695.3
199	5319.7	1982.1	29462.3	NaN	73666.9
200	13.6	2917.2	0	NaN	73666.9
Penalties statistics					
	Penalty 1	Penalty 2	Penalty 3	Penalty 4(Modified std.)	
Average	1495.1	1093.8	9443.8	49303.3	
Standard deviation	1818.6	1745.4	14826.4	22676.0	

To give each penalty a fair and equal consideration, the penalty values are normalized by

$$p_n = \frac{p_i - \bar{p}_i}{s_{p_i}} \quad (5.8)$$

where  $p_n$  is the normalized penalty value,  $p_i$  is the original penalty value,  $\bar{p}_i$  the average of the penalty in question and  $s_{p_i}$  the standard deviation of the penalty. The resulting *normalised* penalties and the sum of the four normalised penalties are presented in Table 5.6.

TABLE 5.6: Normalised penalty value table

Sample	Normalised				Sum
	Penalty 1	Penalty 2	Penalty 3	Penalty 4	
1	-0.6169	0.2565	-0.6370	1.0744	0.0771
2	-0.8103	0.9007	1.5138	1.0744	2.6786
3	-0.5651	-0.6267	-0.6370	-0.7138	-2.5426
...	...	...	...	...	...
199	2.1031	0.5089	1.3502	1.0744	5.0367
200	-0.8147	1.0447	-0.6370	1.0744	0.6675

### 5.3.3 Surrogates

After the 200 sample designs were generated and their corresponding total penalty values determined (Table 5.6), two surrogate modelling methods, RBF and SVR, as described in Section 2.3.2 and 3.2 respectively, were used to generate the knowledge surrogate models. The surrogates were built, tuned and the results compared against each other based on the same sampling plan, which represents the knowledge we have at hand.

#### 5.3.3.1 RBF surrogate, cross validation and visualisation

The RBF method adopted here used a Gaussian basis:

$$\psi(r) = e^{-\frac{r^2}{2\sigma^2}}. \quad (5.9)$$

To estimate the value of  $\sigma$ , a cross validation method is deployed. The 200 designs were divided into 10 random subsets with 20 samples in each subset. A list of 20 logarithmically spaced  $\sigma$ s between  $10^{-3}$  and  $10^0$  was populated as candidates.

For each value of  $\sigma$ , the RBF model was fitted by removing one subset and fitting the model to the remaining, aggregated 9 subsets. A loss function  $L$ , which measures the error between the predictor and the points in the subset that has been set aside, can then be computed by

$$L = \sum_{i=1}^{10} \sum_{j=1}^{20} \left[ y_j^{(i)} - \widehat{f^{(-i)}}(x_j^{(i)}) \right]^2 \quad (5.10)$$

where  $\widehat{f^{(-i)}}$  is the RBF function obtained by removing the  $i^{th}$  subset and  $x_j^{(i)}$  is the  $j^{th}$  training point in the  $i^{th}$  subset.  $\widehat{f^{(-i)}}(x_j^{(i)})$  is the prediction value at  $x_j^{(i)}$  and its difference between the true value  $y_j^{(i)}$  indicates the error between the predictor and the true value. The cross-validation errors are calculated and presented in Table 5.7. As the 200 training points are randomly divided into the 10 subsets, each time the cross validation result could be different. Therefore, three runs were performed. It can be seen

TABLE 5.7: RBF cross validation result

No.	$\sigma$	$L$		
		Run 1	Run 2	Run 3
1	0.0010	1056.88	1056.88	1056.88
2	0.0014	1056.88	1056.88	1056.88
3	0.0021	1056.88	1056.88	1056.88
4	0.0030	1056.88	1056.88	1056.88
5	0.0043	1056.88	1056.88	1056.88
6	0.0062	1056.88	1056.88	1056.88
7	0.0089	1056.88	1056.88	1056.88
8	0.0127	1056.88	1056.88	1056.88
9	0.0183	1056.88	1056.88	1056.880
10	0.0264	1056.87	1056.87	1056.87
11	0.0379	1056.24	1056.23	1056.22
12	0.0546	1040.32	1039.69	1039.99
13	0.0785	923.26	922.76	924.91
14	0.1129	629.49	629.01	636.63
15	0.1624	346.55	333.20	343.32
16	0.2336	237.72	218.50	220.47
17	0.3360	301.42	261.75	236.28
18	0.4833	642.64	579.55	470.38
19	0.6952	1264.68	1239.53	935.00
20	1.0000	2113.82	2017.62	1531.05

from Table 5.7 that the minimum value of  $L$  can always be achieved when  $\sigma = 0.2336$ . Therefore,  $\sigma = 0.2336$  is used in the RBF surrogate.

The RBF surrogate is a function of four design variables ( $x_1$  to  $x_4$ ), and is therefore difficult to visualise. Here a nested dimensions plot see, e.g., [Forrester et al. (2008)] is employed to present the “landscape” of the surrogate. In Figure 5.15,  $x_3$  varies along the horizontal axis of each tile,  $x_4$  along the vertical axes, while the values of  $x_1$  and  $x_2$  can be read off the bottom of each column of tiles and the beginning of each row, respectively.

### 5.3.3.2 SVR surrogate, tuning and visualisation

As an alternative to RBF, a Gaussian kernel based SVR surrogate is also examined. The kernel has the following form:

$$\psi[(x)^{(i)}, (x)^{(j)}] = \exp \left\{ - \frac{[(x)^{(i)} - (x)^{(j)}]^2}{\sigma^2} \right\}. \quad (5.11)$$

The value of  $\sigma$  in Equation 5.11 also needs to be determined. A list of candidates of  $\sigma$  is populated:

$$\sigma = 0.1, 0.2, 0.3, \dots, 0.9, 1. \quad (5.12)$$

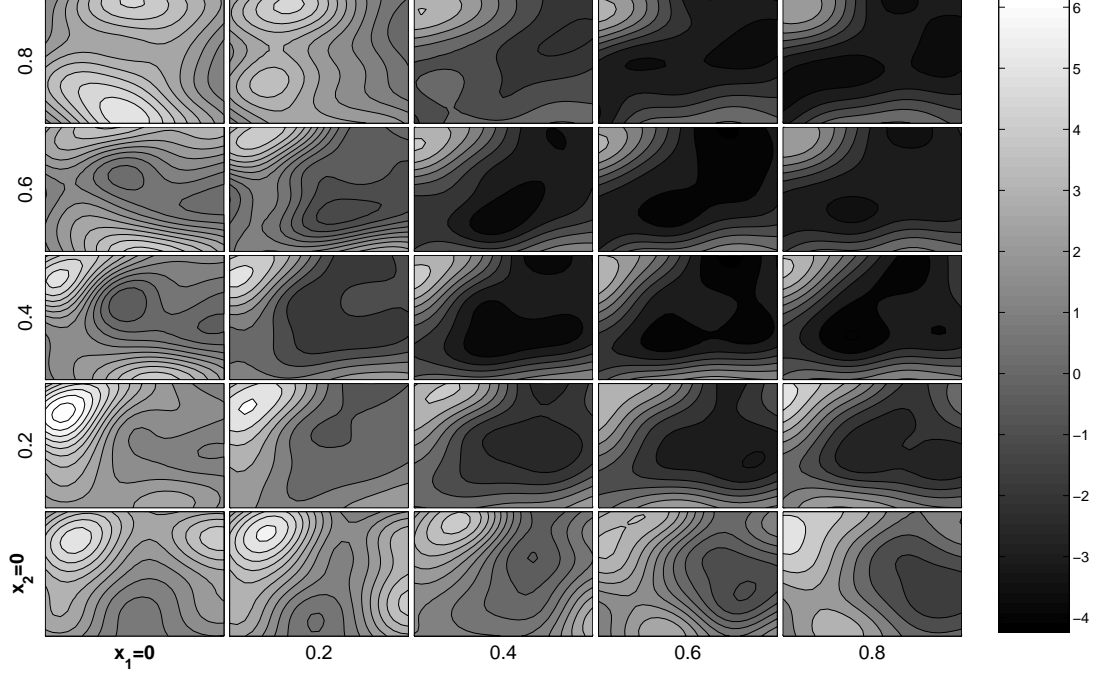


FIGURE 5.15: RBF surrogate landscape

As the SVR model is a regression model, it makes sense to directly compare the predictions against the original sums of penalties (whereas in the case of the RBF surrogate, as it is an interpolating model, the prediction and the training data will always be equal). To determine the value of  $\sigma$  that can best represent the training data, the square of the difference in the training data and SVR prediction are summed for each  $\sigma$  in Equation 5.12. The  $\sigma$  that produces the least error is chosen. Formally, the sum of the error for the  $k^{th}$   $\sigma$  is denoted as

$$L_k^2 = \sum_{i=1}^{200} \left[ y_i - \widehat{f}_{\sigma_k}(\mathbf{x}_i) \right]^2 \quad (5.13)$$

where  $y_i$  is the true value of the penalty and  $\widehat{f}_{\sigma_k}(\mathbf{x}_i)$  is the SVR prediction at the training site  $\mathbf{x}_i$  using  $\sigma_k$ .

The difference in the training data and SVR prediction could be measured by other metrics such as the sum of the absolute values of the difference in the training data and SVR prediction, which is denoted as  $|L_k|$  here:

$$|L_k| = \sum_{i=1}^{200} |y_i - \widehat{f}_{\sigma_k}(\mathbf{x}_i)|. \quad (5.14)$$

The  $\sigma$ s and their corresponding prediction error value  $L_k^2$ s and  $|L_k|$ s ( $k=1, 2, \dots, 9, 10$ ) are presented in Table 5.8. It can be seen from Table 5.8 that when  $\sigma = 0.4$ ,  $L_k^2$  is minimised, when  $\sigma = 0.5$ ,  $|L_k|$  is minimised. Therefore we will use both  $\sigma$ s and compare them in Section 5.3.4.

TABLE 5.8:  $\sigma$  vs. Prediction Errors

$\sigma$	$L_k^2$	$ L_k $
0.1	709.97	286.02
0.2	498.54	225.23
0.3	304.49	162.19
0.4	233.71	<b>149.46</b>
0.5	<b>214.54</b>	151.28
0.6	222.30	156.93
0.7	239.49	165.84
0.8	273.61	179.16
0.9	317.06	195.36

The “landscape” images of the SVR surrogates for both  $\sigma$ s are generated in a way similar to how the RBF surrogate was generated, see Figures 5.16 and 5.17 for the landscape.

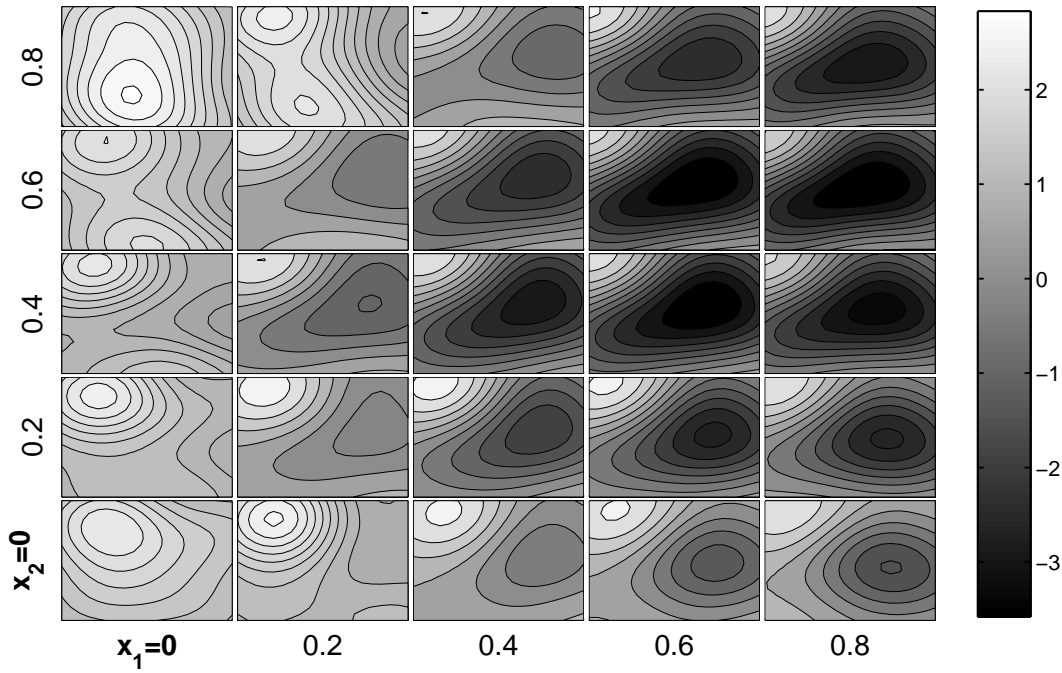


FIGURE 5.16: SVR surrogate landscape ( $\sigma = 0.4$ )

### 5.3.4 Data validation

To find out whether the surrogate models we have obtained can effectively represent the knowledge base and how well they can predict the penalty level for any new designs, 20 new sample designs were generated and their penalties were determined in the same way that those of the 200 designs used to construct the surrogate. The normalisation

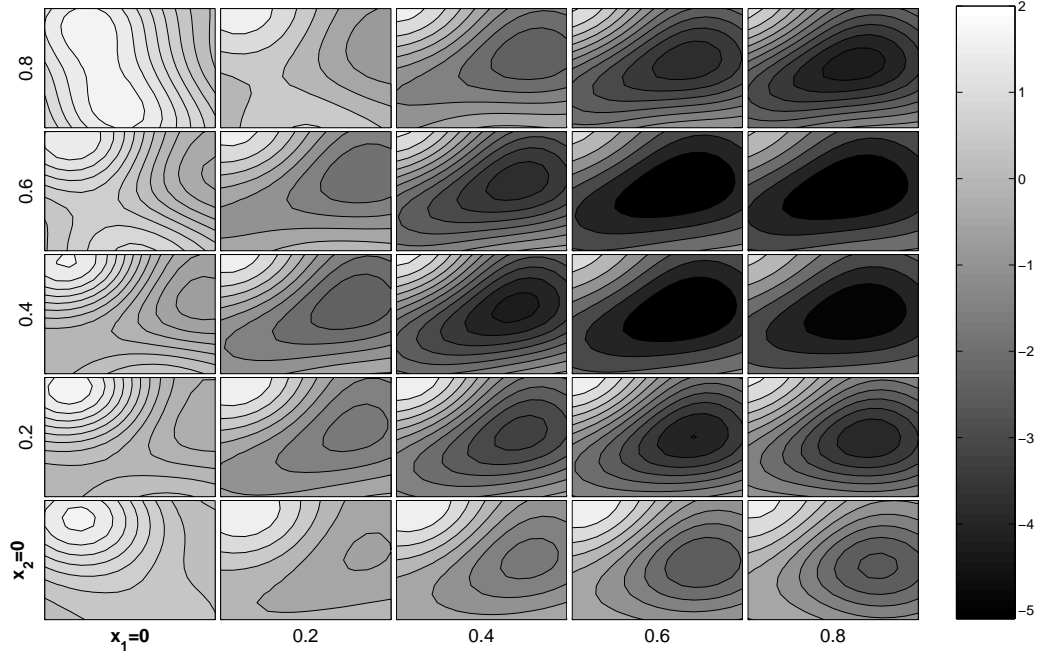


FIGURE 5.17: SVR surrogate landscape ( $\sigma = 0.5$ )

parameters  $p_i$  and  $s_{p_i}$  ( $i = 1, 2, 3, 4$ ) used in Equation 5.8 for the new samples are from Table 5.5. In the same manner, a penalty table was assembled. This time, as three surrogates (one RBF and two SVR models) have been obtained, the normalised penalty sum in the penalty table can be compared against the prediction values from the surrogate models. The 20 new normalised penalty sum, predictions from RBF and SVR models are listed in Table 5.9. To measure the error for each surrogate, the sum of the square of difference  $E_{sqr}$  is calculated for each of the surrogate, which is denoted as  $\hat{f}$  in Equation 5.15.

$$E_{sqr} = \sum_{i=1}^{20} \left[ \hat{f}(\mathbf{x}_i) - y_i \right]^2 . \quad (5.15)$$

For all the surrogate models, the absolute value error,  $E_{abs}$  is also calculated:

$$E_{abs} = \sum_{i=1}^{20} |\hat{f}(\mathbf{x}_i) - y_i| \quad (5.16)$$

The results are listed in Table 5.10.

Two conclusions can be drawn from Table 5.10.

1. After each surrogate was tuned to give its best performance, both SVR surrogates give better prediction results than the RBF surrogate does, no matter which metric



TABLE 5.9: 20 new designs, their true penalties, RBF and SVR predictions

Sample	True penalty	RBF	SVR( $\sigma = 0.4$ )	SVR( $\sigma = 0.5$ )
1	-0.965	0.036	-3.488	-1.847
2	-0.602	-0.009	-0.481	-0.288
3	1.689	-0.235	2.092	0.662
4	-1.830	0.000	-1.921	-0.481
5	-2.125	-0.004	0.551	0.262
6	-2.933	0.019	-2.548	-1.173
7	-2.968	-0.062	-1.132	-0.509
8	-0.804	0.094	1.167	0.631
9	1.299	-0.014	2.284	0.829
10	-0.624	0.000	-0.673	-0.266
11	-0.845	0.001	-0.373	-0.302
12	0.601	-0.004	0.849	0.325
13	3.534	-0.002	2.612	0.547
14	-0.434	-0.193	1.002	0.216
15	-1.581	0.000	0.178	-0.247
16	-1.329	-0.037	0.499	0.622
17	-0.335	-0.002	-0.520	-0.156
18	-0.478	0.113	0.695	0.312
19	3.910	0.000	2.081	0.339
20	0.099	-0.025	2.200	0.715

TABLE 5.10: Comparison of prediction errors for different surrogates

	$E_{sqr}$	$E_{sqr}$
RBF	66.6	29.22
SVR ( $\sigma = 0.4$ )	40.9	23.0
SVR ( $\sigma = 0.5$ )	50.1	25.3

is used to measure the prediction error (Equation 5.15 or Equation 5.16). Therefore the author conclude that SVR is better at predicting new values for this particular problem, and therefore better at representing the knowledge base than RBF in this case study.

2. The  $L_k^2$  metric is better at selecting the  $\sigma$  values for SVR surrogate, as the SVR surrogate with  $\sigma = 0.4$ , which is chosen by minimising  $L_k^2$ , predicts better than the one with  $\sigma = 0.5$ , no matter which metric is used to measure the prediction error.

## 5.4 Repair

To investigate the proposed method's ability to evaluate an unknown design candidate and provide repair suggestions, a design with multiple flaws is picked (Figure 5.18). The intake face of the design is slightly submerged in the fuselage and the intake itself is

snaky and interferes with the aft bulkhead. The design is so convoluted that it becomes unphysical, to the extent that it fails to be meshed. The design variables of the geometry

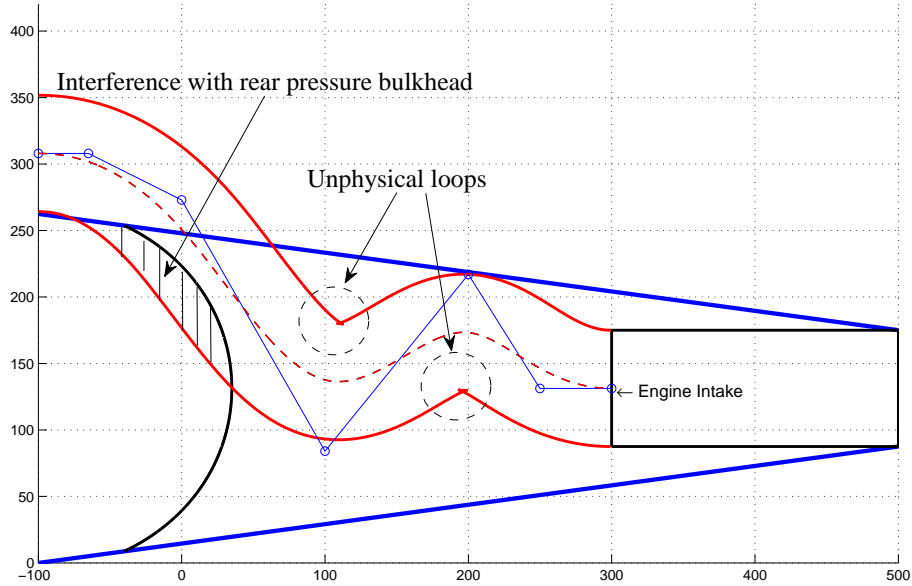


FIGURE 5.18: A design candidate with multiple flaws

shown in Figure 5.18 are

$$\mathbf{x} = [0.45, 0.35, 0.05, 0.65].$$

The predicted penalty values based on different surrogates were first calculated and are listed in Table 5.11. These values are used to determine a starting value for a list of feasibility thresholds — any feasibility threshold that is larger than the predicted penalty is meaningless. The true penalty was also calculated and listed in Table 5.11 for comparison purpose. The feasibility thresholds used for the repair and results based

TABLE 5.11: Predicted penalty values	
Surrogate	Predicted Penalty
RBF	2.425
SVR ( $\sigma = 0.4$ )	0.9398
SVR ( $\sigma = 0.5$ )	0.5615
True Penalty	2.455

on the RBF surrogate are listed in Table 5.12 alongside the resulting geometries.

Table 5.12: Design alternatives based on the RBF surrogate

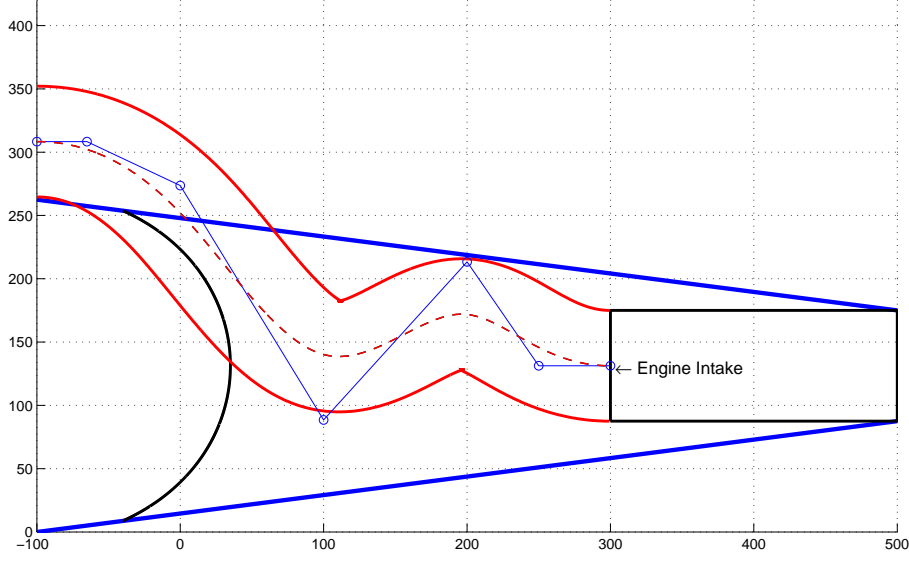
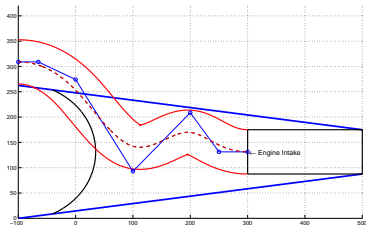
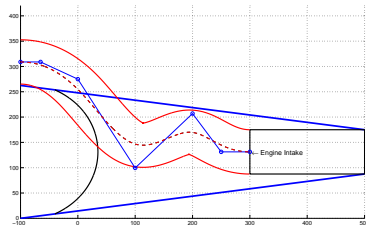
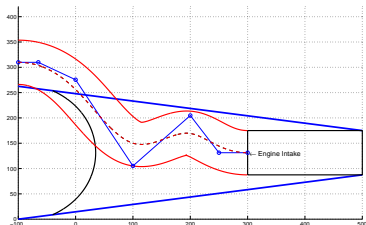
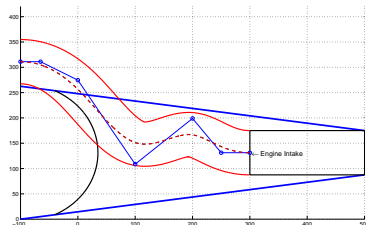
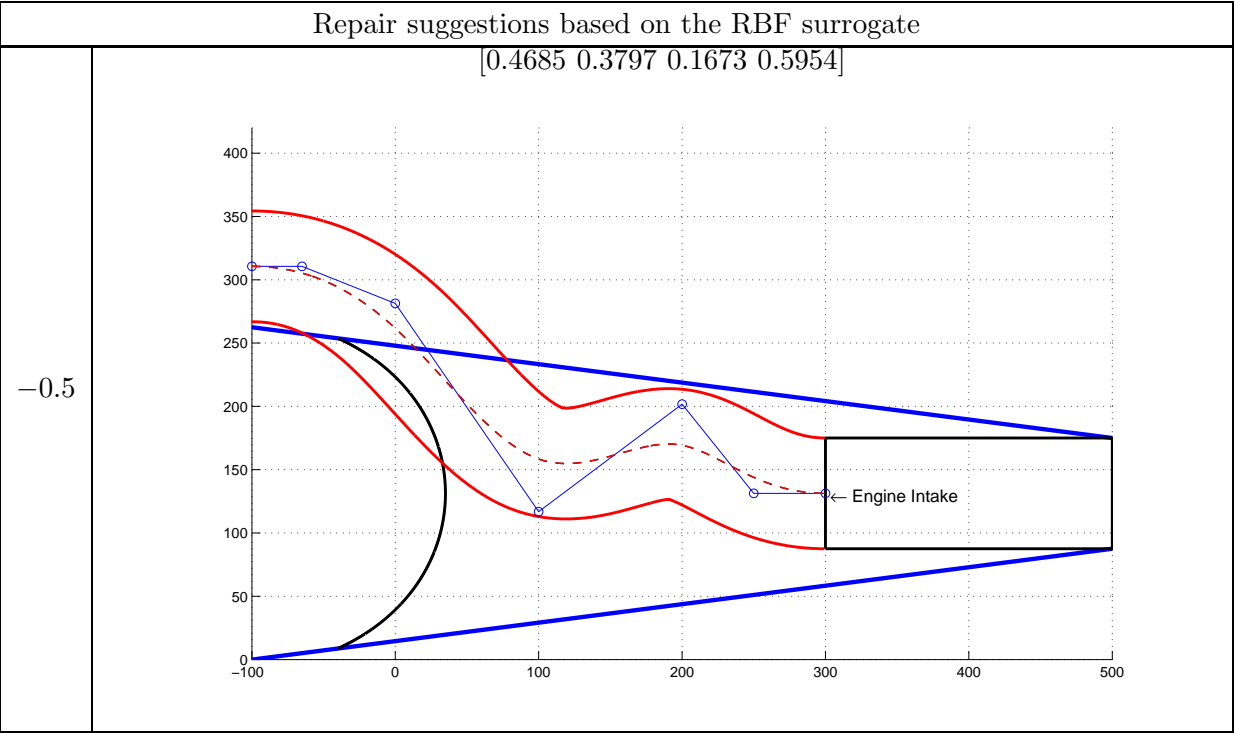
$p_{th}$	Repair alternative suggestion [0.4526 0.3523 0.0661 0.6370]				
2.0					
$p_{th}$	Repair alternative suggestion [0.4571 0.3543 0.0816 0.6201]		$p_{th}$	Repair alternative suggestion [0.4563 0.3571 0.1055 0.6140]	
1.5			1.0		
0.5			0		

Table 5.12: (continued)



The feasibility thresholds used for the repair and results based on the SVR surrogate ( $\sigma = 0.4$ ) are listed in Table 5.13, again with the resulting geometries.

Table 5.13: Repair suggestions based on the SVR surrogate  
( $\sigma = 0.4$ )

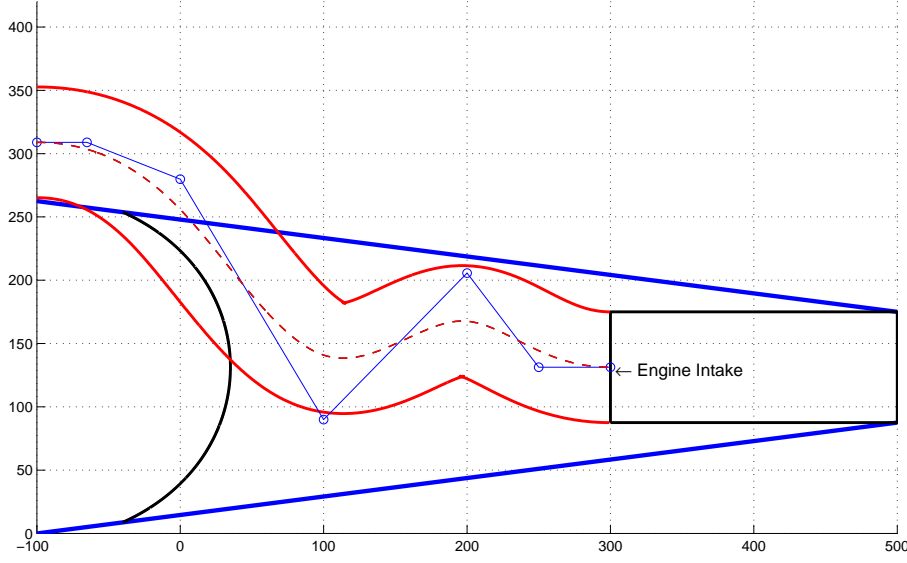
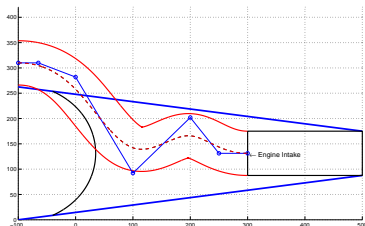
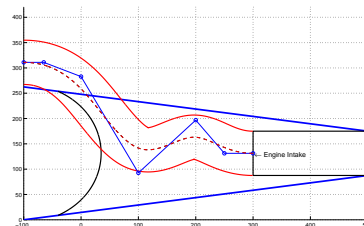
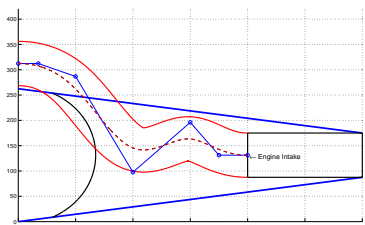
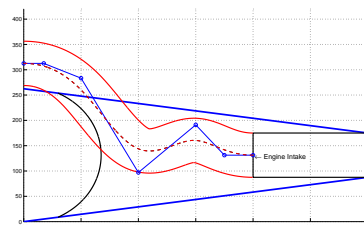
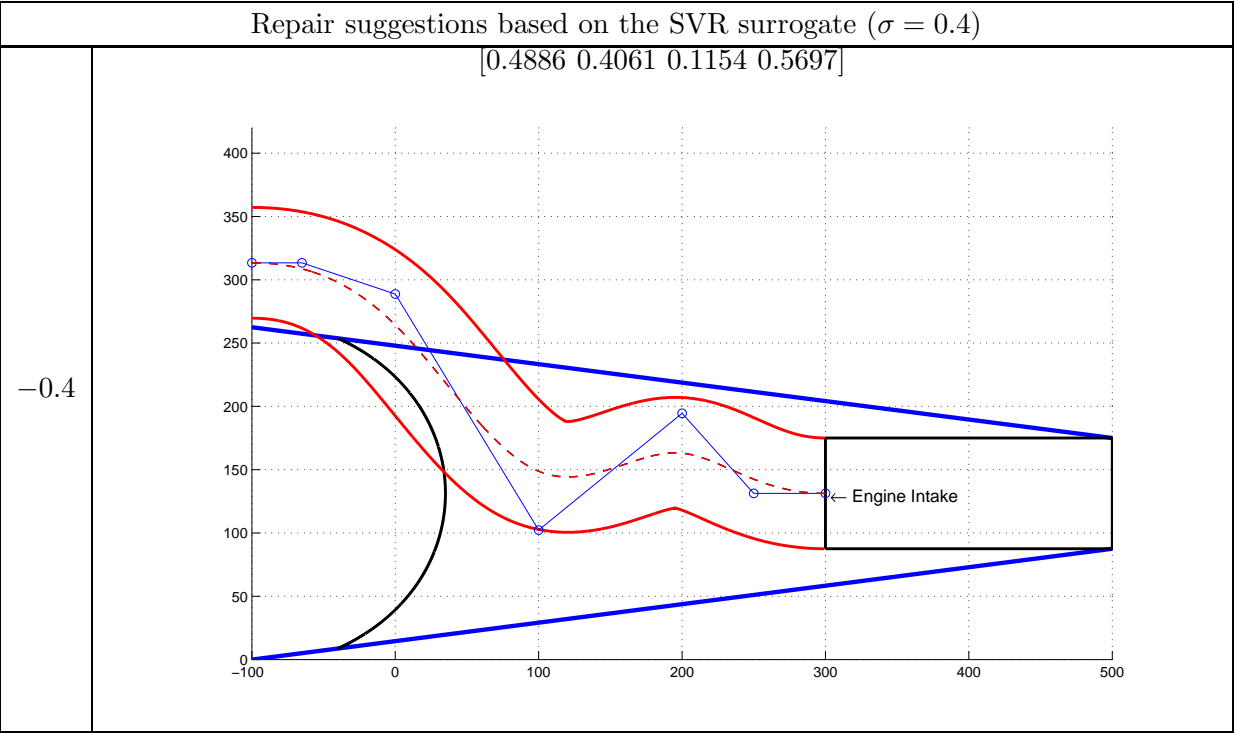
$p_{th}$	Repair alternative suggestion [0.4566 0.3741 0.0712 0.6094]		
0.4			
$p_{th}$	Repair alternative suggestion [0.4638 0.3818 0.0794 0.5964]	$p_{th}$	Repair alternative suggestion [0.4715 0.3850 0.0808 0.5781]
0.2		0	
-0.2		-0.3	

Table 5.13: (continued)



The feasibility thresholds used for the repair and results based on the SVR surrogate ( $\sigma = 0.5$ ) are listed in Table 5.14.

Table 5.14: Repair suggestions based on the SVR surrogate  
( $\sigma = 0.5$ )

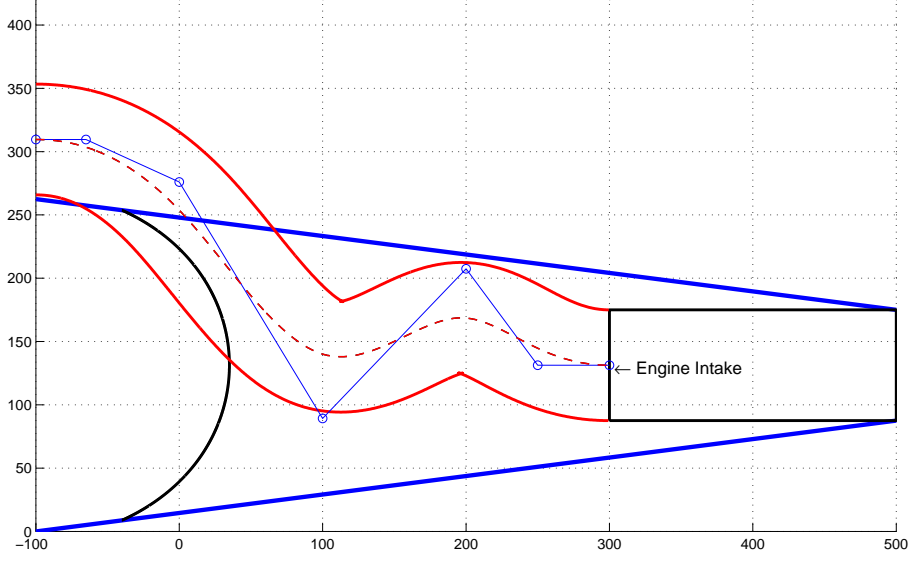
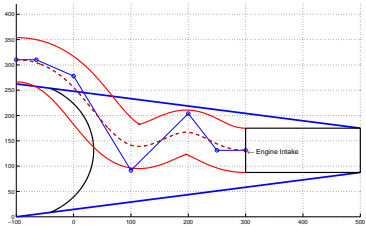
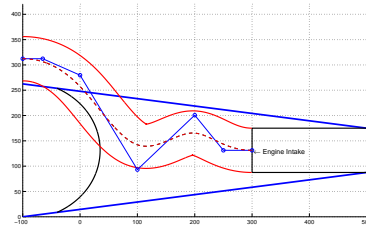
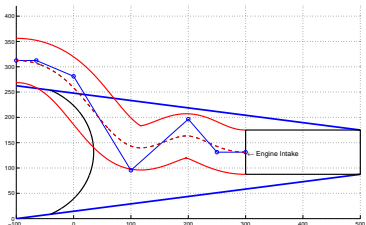
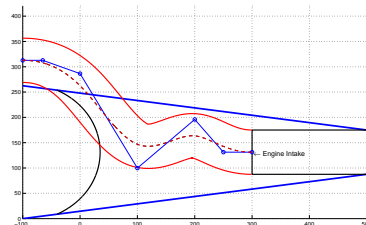
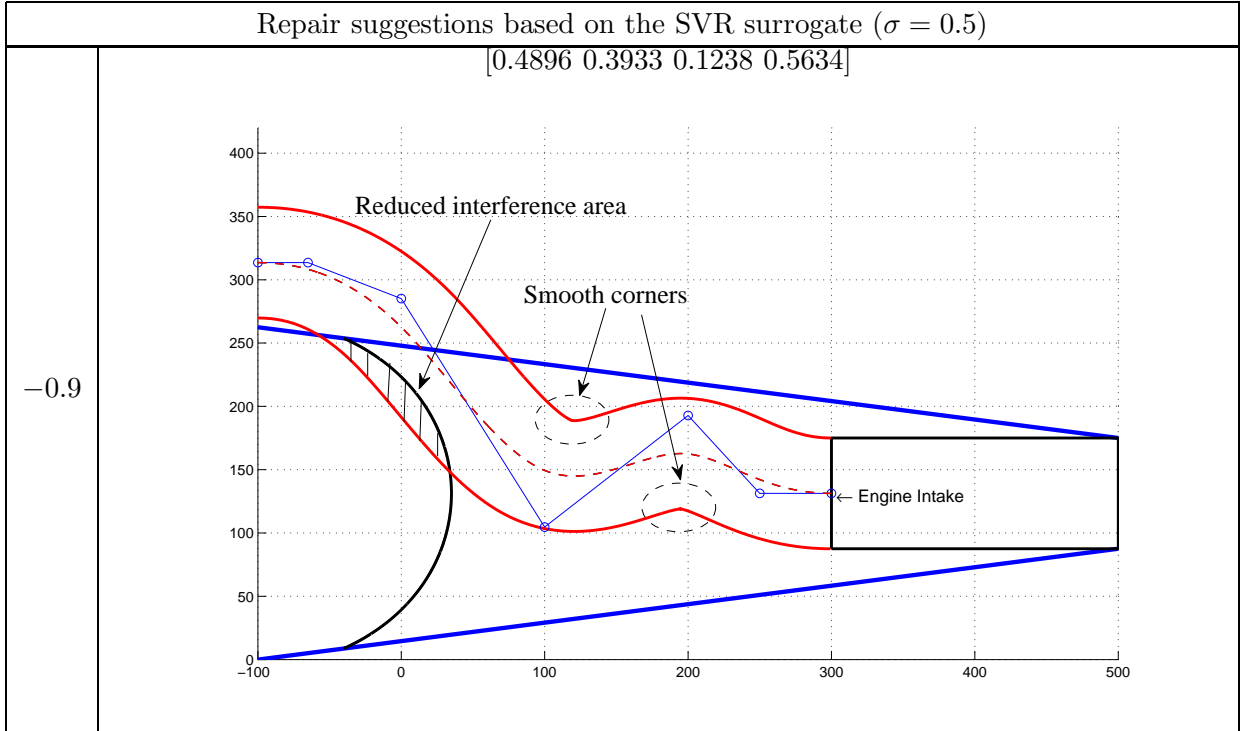
$p_{th}$	Repair alternative suggestion [0.4613 0.3606 0.0687 0.6158]			
0.1				
$p_{th}$	Repair alternative suggestion [0.4656 0.3679 0.0777 0.6031]	$p_{th}$	Repair alternative suggestion [0.4803 0.3740 0.0828 0.5920]	
-0.1		-0.3		
-0.5		-0.7		

Table 5.14: (continued)



As expected, the repair suggestions all look similar to the original design (Figure 5.18), which is a sign of the preservation of the original design intent. However subtle changes do occur, which correct the meshing failure and improve the feasibility of the geometry. For example, if compare Figure 5.18 and the last figure in Table 5.14, it is not difficult to notice that:

1. the unphysical loop and the first order discontinuities, which occurred on the intake wall between  $x = 100$  and  $x = 200$  have been eliminated;
2. the area of interference with the rear pressure bulkhead has been gradually diminished and eventually disappears altogether.

The unphysical loops were responsible for the mesh failure. After they have been replaced by the smooth corners, the geometry became meshable. Furthermore, the suggested repair alternative has a reduced area of interference with the bulkhead, due to the knowledge that dictates that such interference is an unfavourable feature (Section 4.3.3).

From Table 5.12 and Table 5.13, it could be observed that similar repair results have been achieved with RBF surrogate model and a SVR surrogate with a different  $\sigma$ . The repaired geometries become gradually more feasible, as the feasibility threshold increases. Therefore, we conclude that the knowledge that was set up in Section 4.3 has been successfully incorporated in the knowledge surrogate, and the automatic repair system is effective.



## 5.5 Surrogate update

In this section, the author demonstrates how an existing surrogate can be updated using new information and investigates the effect of adding more training points to an existing surrogate model on geometry repair. The SVR surrogate with a preset  $\sigma = 0.5$  is used for demonstration in this process. Four badly designed geometries were manually picked as the starting points for repair. For each badly designed geometry, its feasibility index is firstly determined by the current SVR surrogate. Then eight repair alternatives are determined by the method described in Section 3.4. After the repaired geometries were determined for the four geometries, the true feasibility values for these 32 repair alternatives were calculated by the method described in Section 4.3. These 32 sets of data were added to the original 200 training data. Another 20 sets of data have been determined in Section 5.3.4 as a validation data set. We then reused these data and added them to the existing training data pool.

The SVR surrogate is updated with the 252 ( $200 + 20 + 32$ ) training data. In terms of the computing budget, this updating process is very cheap — the training of the surrogate takes only a few minutes. The updated SVR surrogate is used for the repair of the same four badly designed geometries again. In this way another 32 repair alternatives were generated, the feasibility values calculated and the data were used to update the surrogate (284 training points altogether). The update process is repeated one more time with 316 training points, the last 32 of which were from the repair alternatives generated with the SVR surrogate training by 284 points.

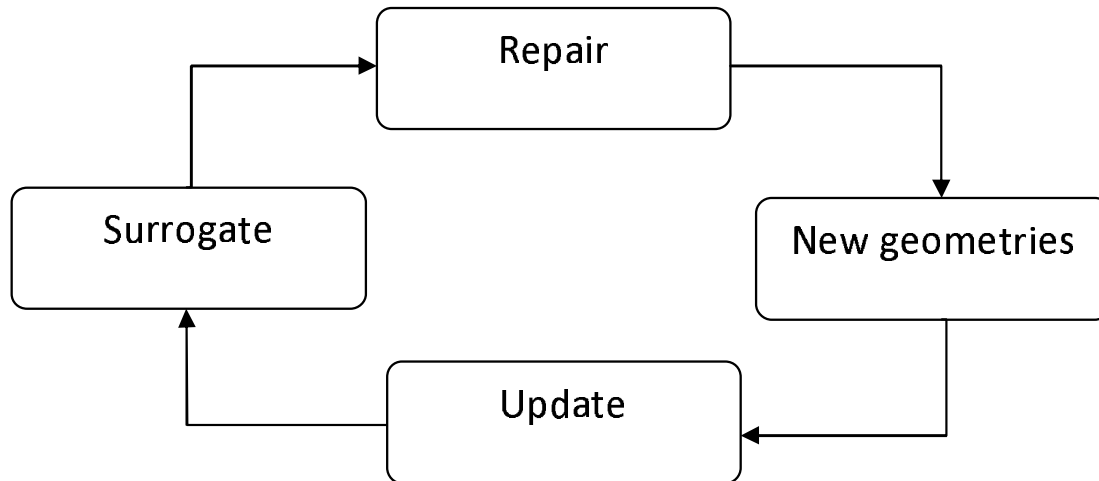


FIGURE 5.19: Illustration of the surrogate model update process

Altogether 96 update points were generated. To find out the effect of adding more training points to the existing surrogate, the true penalty values and feasibility predictions from the SVR surrogate of 200 points, 252 points, 284 points and 316 points were

calculated. Then the absolute value of the difference between true penalty values and predictions were calculated. Some of the results are presented in Table 5.15 and 5.16.

TABLE 5.15: Update training points' true penalties and SVR predictions

No.of sample	True Penalty	SVR(200)	SVR(252)	SVR(284)	SVR(316)
1	1.501	0.910	0.997	1.068	1.118
2	0.749	0.544	0.592	0.646	0.681
3	0.183	0.181	0.206	0.244	0.268
...	...	...	...	...	...
94	-1.318	-0.678	-0.898	-0.933	-1.115
95	-1.872	-1.298	-1.445	-1.502	-1.669
96	-2.342	-1.883	-1.979	-2.047	-2.220

TABLE 5.16: Comparison of surrogates between updates

No. of sample	True Penalty	Diff.  (200)	Diff. (252)	Diff.  (284)	Diff.  (316)
1	1.501	0.591	0.504	0.433	0.383
2	0.749	0.205	0.158	0.103	0.068
3	0.183	0.001	0.023	0.062	0.085
...	...	...	...	...	...
94	-1.318	0.640	0.420	0.386	0.204
95	-1.872	0.574	0.427	0.369	0.203
96	-2.342	0.459	0.363	0.295	0.123
Average difference		0.683	0.638	0.579	0.531

The average difference listed in the last row of Table 5.16 measures the fidelity of the knowledge surrogate model. It can be seen that as the surrogate is updated, the differences reduce correspondingly. Therefore, the update is helpful in obtaining a higher fidelity surrogate that better represents knowledge about the design.



## Chapter 6

# 3D Case Study: Repair of a Turbine Blade Geometry

In this chapter, the author endeavours to extend the idea of geometry repair to a more complicated three-dimensional CAD model. The repair of a real-life model would be potentially beneficial and of commercial interest to industry, such as the author's industrial sponsor Rolls-Royce, as a means of smoothing the design workflow. A possible 3D case may be the Rolls-Royce blade and core model that other members of the author's research group are investigating from a different perspective (life prediction, etc.). The cores are air ducts inside the blade, filled with cold circulating air while in service. The air cools the blade down, therefore prolongs the life of the blade. The model is quite complex and can easily fail in a variety of ways if the parameterisation goes wrong, for example, the cores can penetrate the blade surface or overlap with each other, which could lead to failed designs. If the CAD model fails, any subsequent analysis will become impossible. Therefore, it is potentially useful if a knowledge base can be set up and attached to the CAD model, in order to automatically examine the quality of the model and provide helpful advice which can lead to a valid and better design.

In this chapter, the author sets up a repair system for an 3D turbine blade, based on the principles of the geometry repair system described in Chapter 3 and tested in Chapter 4. This system can fit in an optimisation framework, which helps the automation of the optimisation.

### 6.1 Overview of a turbine blade design problem

Turbine blades are amongst the main components of a typical gas turbine. They are designed to generate power by translating circumferential aerodynamic forces on the aerofoil to the rotating disc. The blades are of a cambered aerofoil shape, designed to

extract energy from the high temperature, high pressure gas produced by the combustor. The blades glow red-hot when the engine is running, yet at this condition they must still be strong enough to carry high centrifugal loads due to their rotation and the bending load due to the gas stream.

The blades rotate in the casings with a typical tip speed of 460 m/s [Rolls-Royce (2005)]. At this speed, the power output of a single civil HP blade is ten times higher than that of a small family car and the force transmitted into the disc by each blade at redline speed is approximately 18 tonnes.

The blade's cross-section design is governed by the permitted stress in the material used and by the size of any core passages required for cooling purposes. The hottest running blades are cast in a high-temperature nickel alloy and are often coated in a ceramic thermal barrier coating on their aerofoils and platforms. High operating temperatures dictate the need to internally cool the HP blades with cooling air flowing through a complex internal channel system before exiting through rows of cooling holes. Cooling flow is regulated very carefully to minimise the detrimental effect on turbine performance. The following picture shows a high pressure turbine blade with internal cooling system<sup>1</sup>.

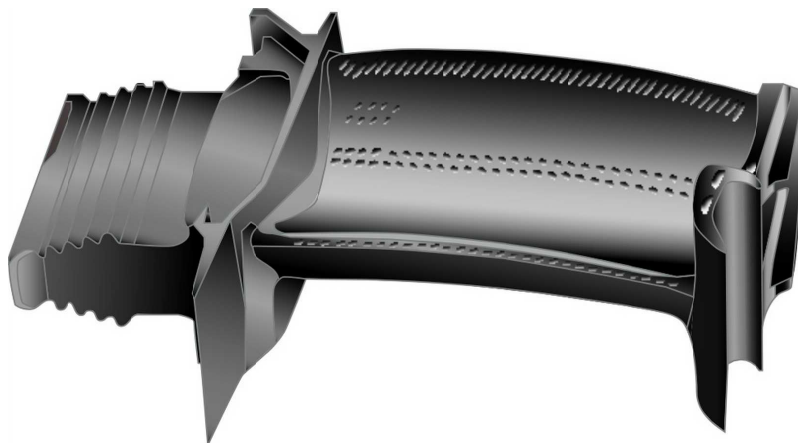


FIGURE 6.1: A turbine blade with cooling holes for film cooling

As turbine designs progress through each new engine project, the basic design and operating principles remain the same as those used in the very earliest of turbine designs. Today, however, modern market requirements combined with reduced timescales add pressure to the design and development programme. The focus of investment and development on the latest products is channelled towards even more demanding targets in turbine performance and efficiency—together with reductions in fuel burn, unit cost and engine weight [Rolls-Royce (2005)].

---

<sup>1</sup>This file is made available by Tomeasy from the Wikimedia Commons, a freely licensed media file repository. The file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license. The original link is <http://en.wikipedia.org/wiki/File:GaTurbineBlade.svg>.

## 6.2 An automated turbine blade design and analysis framework

The modern design practice of a turbine blade involves parameterisation, automated geometry generation, analysis and optimisation. In this section, the author set up such a design cycle. Due to the complexity of the feasible design space, the automated optimisation cycle can be interrupted. A repair framework is set up to ensure the continuity of the design workflow, and capture possible optimal designs which may lie at the edge of feasible design space. The automated framework includes CAD model setup, preprocessing, stress analysis and the use of surrogate modelling for predicting the quality of untested geometries and repair the ones that fall below the quality threshold.

### 6.2.1 CAD model setup and parameterisation

First a simplified turbine model is set up. The topology of the model is predefined: three cooling holes are present in the blade 3D model. The blade outline is defined by four splines, the shape of which similar to an airfoil. The exact shape of the blade is determined by 12 variables altogether. The first six variables control the positions of the three cooling holes: the first two define the position of the leading hole; the third and fourth middle hole; the fifth and sixth trailing hole. The next group of six variables defines direction in which the blade cross-section is extruded. The first trio in the group defines the extrusion direction of the cross-section, leading hole and trailing hole. The second trio defines the extrusion direction of the middle cooling hole. The trailing edge of the blade is often the hottest and therefore endures higher temperature and thermal stress. The temperature of the trailing edge could be a bottleneck of the blade design. Therefore the cooling hole which is closest to the trailing edge is of special importance. The last four design variables are used to control the exact shape of the trailing hole.

Variables	Definitions
$x_1$ and $x_2$	Horizontal and vertical position of the front cooling hole
$x_3$ and $x_4$	Horizontal and vertical position of the middle cooling hole
$x_5$ and $x_6$	Horizontal and vertical position of the trailing cooling hole
$x_7, x_8$ and $x_9$	Extrusion direction of the blade contour, front and trailing hole
$x_{10}, x_{11}$ and $x_{12}$	Extrusion direction of the middle cooling hole
$x_{13}$ and $x_{14}$	Trailing hole upper contour point
$x_{15}$ and $x_{16}$	Trailing hole lower contour point

TABLE 6.1: Blade variable definitions

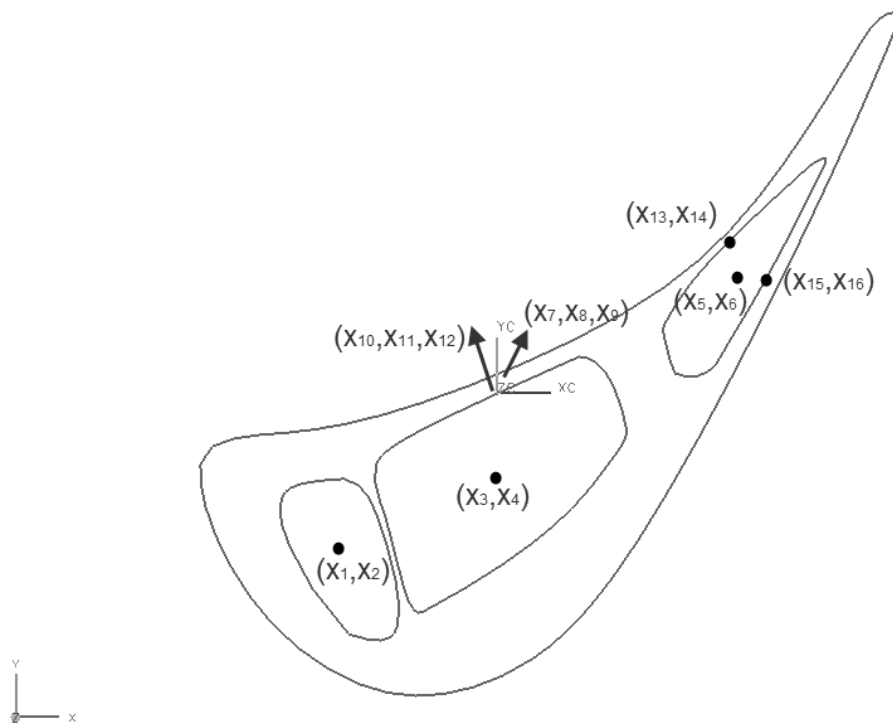


FIGURE 6.2: Turbine Blade cross-section view with illustrations of the design variables

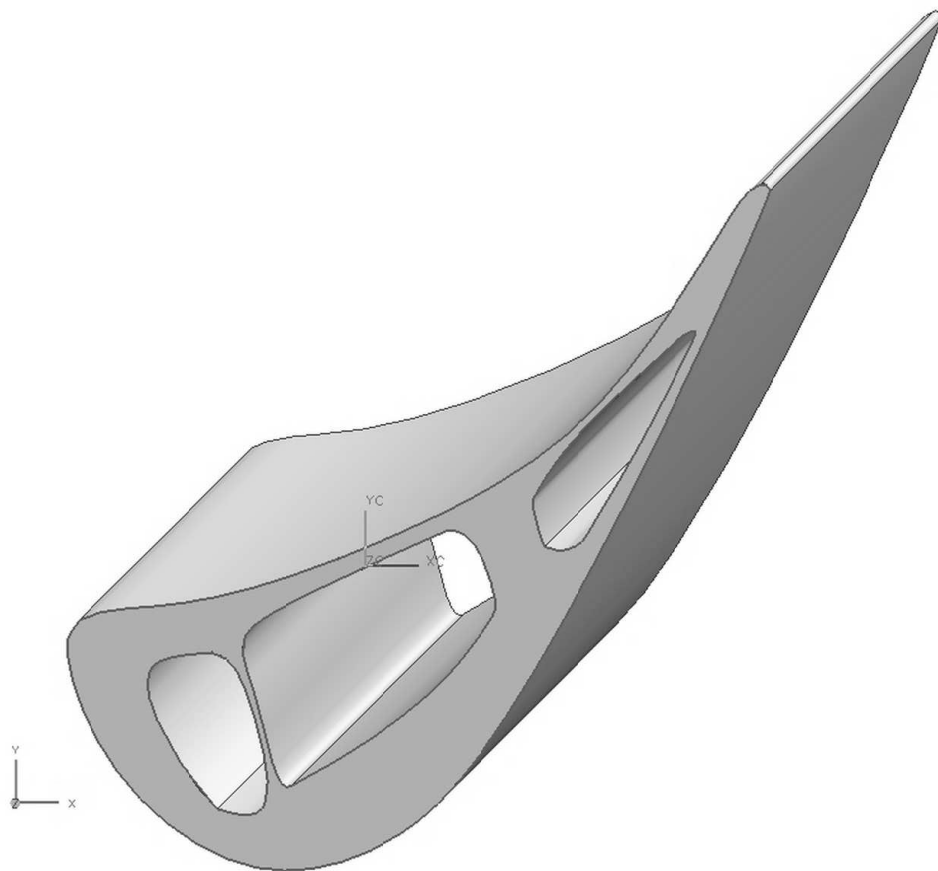


FIGURE 6.3: Turbine blade shaded view

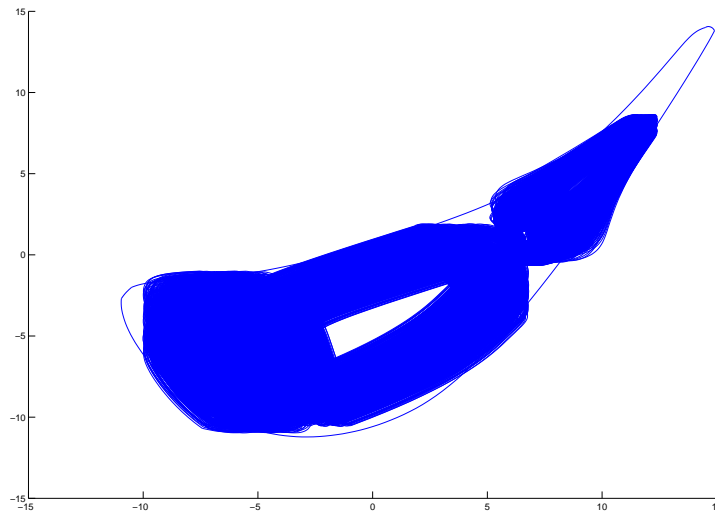


FIGURE 6.4: Possible blade cooling hole position contours

### 6.2.1.1 Geometry generator developed in NX Open

The CAD model is generated by NX Open C. NX Open is a collection of Application Programming Interface (API) toolkits that allow for automating complex and repetitive tasks. NX Open also allows for the flexible integration of diverse third party and NX applications, sharing data on different computer platforms, from different locations using heterogeneous networks, and even across the Internet [Sie (2008)]. The NX Open API provides an open architecture which can be utilised by third parties, customers, and in-house users for creating and integrating custom software applications.

The geometry is generated by the following steps. Firstly the 12 design variables are read into memory from a file that contains design variables. The variable file is generated by a MATLAB programme. A sample variable file is listed in Appendix A.8.1. The variables are read into the memory by a C program, which is listed in Appendix A.8.2.

After the design variables are successfully loaded, they are passed to the API functions that generates the splines and extrusions. The C file that generates the 3D section of the turbine blade is listed in Appendix A.8.3

The above-mentioned C files are compiled into a .dll file, which can be executed within NX interactively. Furthermore, an executable file is compiled, which can be called by a Windows batch file, which is listed in Appendix A.8.4. In this way the geometry can be automatically generated. The batch file requires two Windows environment variables, UGILBASE\_DIR and UGILROOT\_DIR, to be set up and pointed to the NX installation folder path.



There is a minimum blade wall thickness requirement in manufacturing. A guideline thickness is 0.5 millimetres. Designs with blade wall thinner than this value would be unsuitable for manufacturing. The geometry generator can measure distance between the cooling holes, and between the cooling holes and the blade walls. The distances

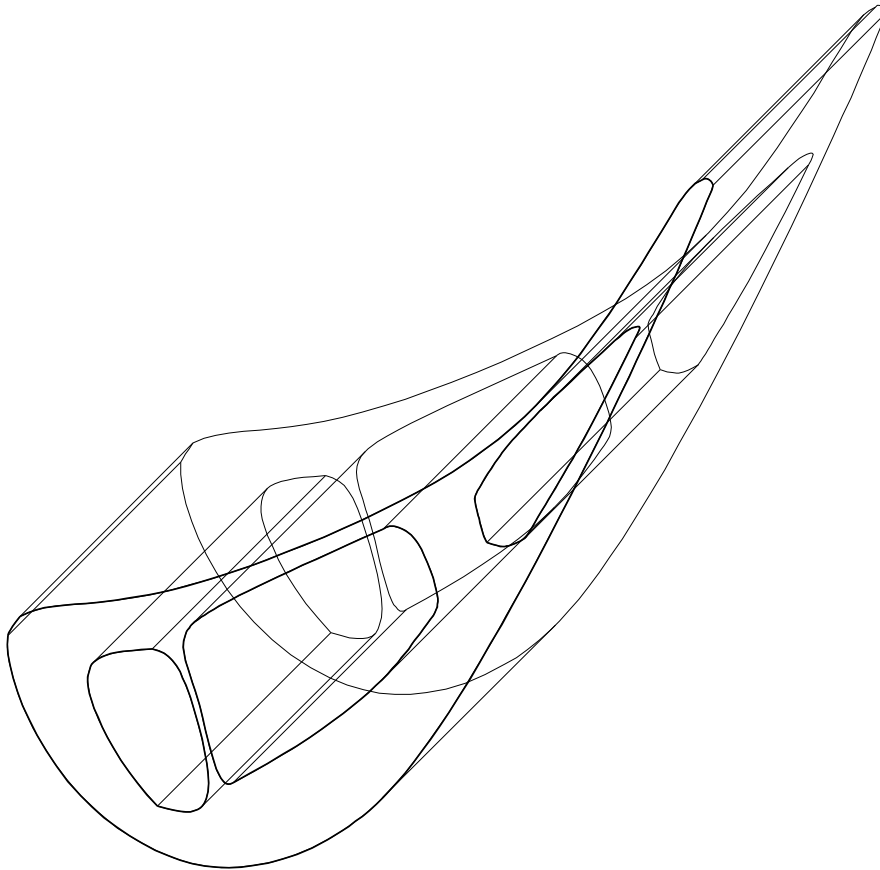


FIGURE 6.5: Turbine blade wireframe view

### 6.2.2 Preprocessing

To facilitate stress analysis, the native NX geometry is firstly exported into IGES format. The NX command line interface allows the translation of files without using the interactive menus. The NX IGES translator is initiated by IGES.cmd, a Windows batch script listed in Appendix A.8.5. The output IGES file name is the same as the input file name, except with the appropriate file extension.

In this research, the Rolls-Royce developed finite element analysis application SC03 is used for thermal stress analysis. The IGES file is translated by CADfix<sup>2</sup> to SC03

<sup>2</sup>A CAD translation software by ITI TranscenData, see [www.cadfix.com](http://www.cadfix.com) for more details

acceptable database file (pm file). This step is automated by calling CADfix in batch mode (A.8.6).

### 6.2.3 Stress analysis

Turbine blades endure high temperatures in working conditions. It is important to predict the stress in blade structure subjected to thermal loads. The von Mises yield criterion is a common yield criteria for ductile materials. Calculation for von Mises stress is readily available in SC03.

#### 6.2.3.1 Von Mises Stress and yield criterion

Von Mises stress is derived from the von Mises yield criterion, which states that yielding begins when the distortional strain energy density at a point equals the distortional strain energy density at yield in uniaxial tension [Boresi and Schmidt (2003)]. The distortional strain energy density  $U_D$  is defined as

$$U_D = \frac{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_1 - \sigma_3)^2}{12G} \quad (6.1)$$

where  $\sigma_1$ ,  $\sigma_2$  and  $\sigma_3$  are principle stress, and  $G$  the shear modulus defined as

$$G = \frac{E}{2(1 + \nu)}. \quad (6.2)$$

The distortional energy density  $U_D$  can be written in terms of the second deviator stress invariant  $J_2$  as

$$U_D = \frac{1}{2G} J_2 \quad (6.3)$$

where

$$J_2 = \frac{1}{6} [(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_1 - \sigma_3)^2]. \quad (6.4)$$

At yield in uniaxial tension or compression,  $\sigma_1 = \pm Y$ ,  $\sigma_2 = \sigma_3 = 0$ . Then

$$J_2 = \frac{1}{3} Y^2 \quad (6.5)$$

Therefore, by Equation 6.4 and 6.5, the yield function for the von Mises criterion can be written as

$$f = \frac{1}{6} [(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_1 - \sigma_3)^2] - \frac{1}{3} Y^2. \quad (6.6)$$

It is convenient to define an equivalent tensile stress or von Mises stress,  $\sigma_v$ , as

$$\sigma_v = \sqrt{\frac{1}{2} [(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_1 - \sigma_3)^2]} = \sqrt{3J_2} \quad (6.7)$$

By Equation 6.7, Equation 6.6 can be written in a more compact form as

$$f = \sigma_v^2 - Y^2 \quad (6.8)$$

The von Mises yield function is continuously differentiable, and is preferred in plasticity studies. There are other yield criteria, including the maximum shear-stress (Tresca) criterion [Boresi and Schmidt (2003)]. No single yield criterion has been established that accurately predicts yielding for all materials. The difference between these criteria are beyond the scope of this thesis.

### 6.2.3.2 Analysis in SC03

SC03 is an automatic analysis system created by Rolls-Royce to allow fully integrated stress, displacement, thermal and vibration analysis to be undertaken at any stage in the design cycle. It was originally designed to meet the aero engine manufacturer's in-house analysis needs.

In this work SC03 is used to mesh the blade geometry and predicts thermal stress and temperature of the blade in simulated working conditions. The process is automated by calling SC03 in batch mode by the script listed in Appendix A.8.7. In the script, an SC03 executable file is called (Appendix A.8.8). This executable file controls SC03 to do the following in turn:

- reading in the pm file;
- setting up working condition and constraint;
- specifying material;
- meshing;
- analysis and output.

Firstly the database file is loaded into SC03. Then a blade material thermo-elastic property data file is loaded (An abridged version of the material property data file is shown in Appendix A.8.9). Then a boundary conditions file is loaded. The file is generated by firstly setting up boundary conditions interactively in SC03, and then saving as a separate file that can be reloaded. The boundary condition file specifies that the front and back flat surfaces as fixed. The convex face endures very hot air when operating. It is assigned as a convecting zone with an air temperature of 1500K and a pressure of 1MPa. The concave face is exposed to relatively cooler air but higher pressure. It is assigned as a convection zone with an air temperature of 1300K and a pressure of 1.25MPa. The cooling holes supply cool air to the blade to keep its

temperature down. They are assigned as convecting zones with air temperatures of 1000K and pressures of 1.5MPa.

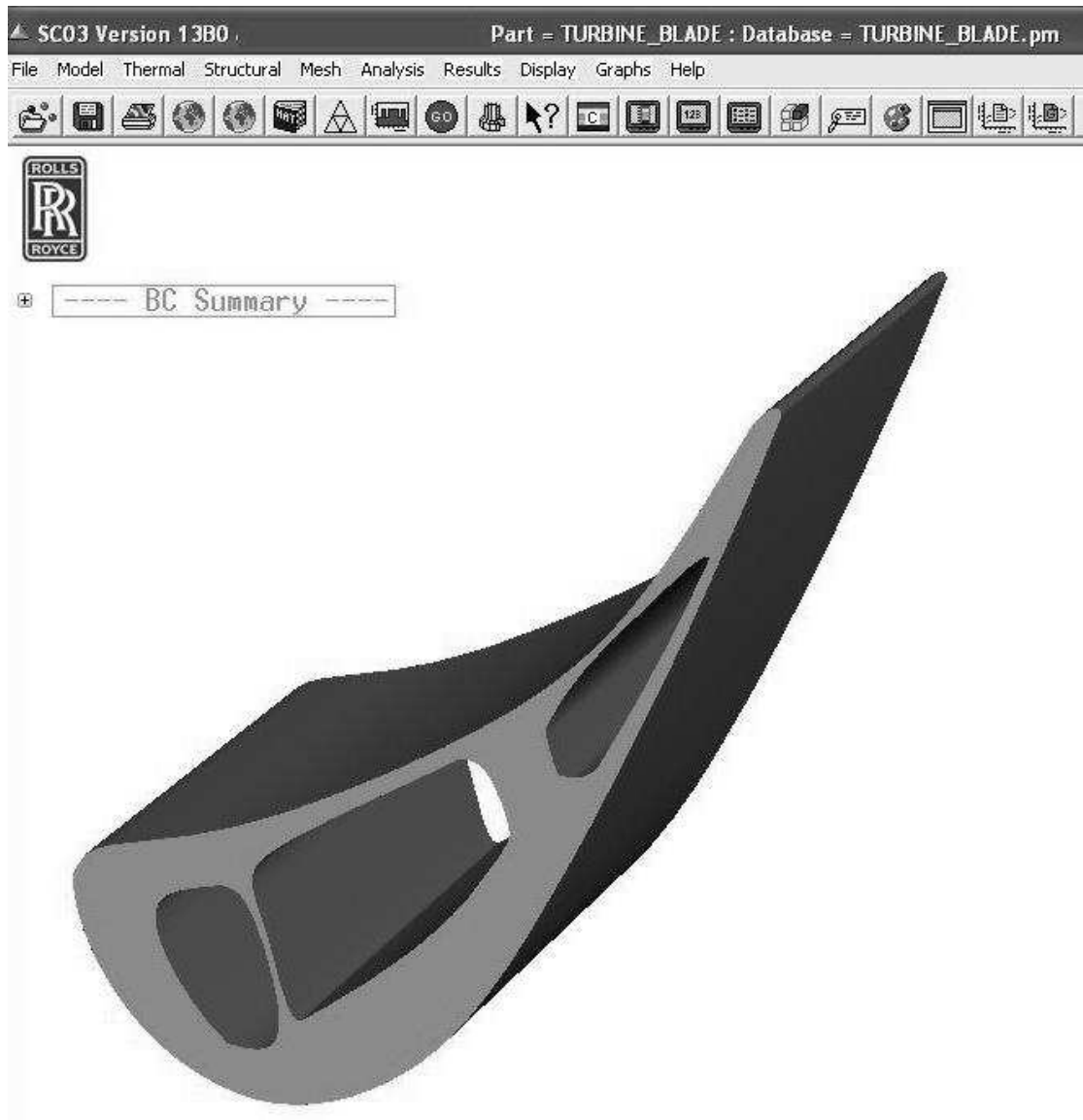


FIGURE 6.6: Blade model imported into SC03

Before meshing, local mesh controls are applied to the blade surfaces. Surface mesh element sizes are specified depending on the local curvature. The convex and concave surface where the curvature is relatively low has an average element size of  $2 \times 10^{-3}$ . Other surfaces, including the cooling holes has an average element size of  $1.5 \times 10^{-3}$ .

If the meshing is successful, a thermo-stress steady state analysis is performed to determine the steady state temperature and von Mises stress of the structure. The results are exported in a SC03 log file first, and the maximum temperature, maximum von Mises stress and largest mesh distortion are extracted by a MATLAB script (Appendix A.8.11).

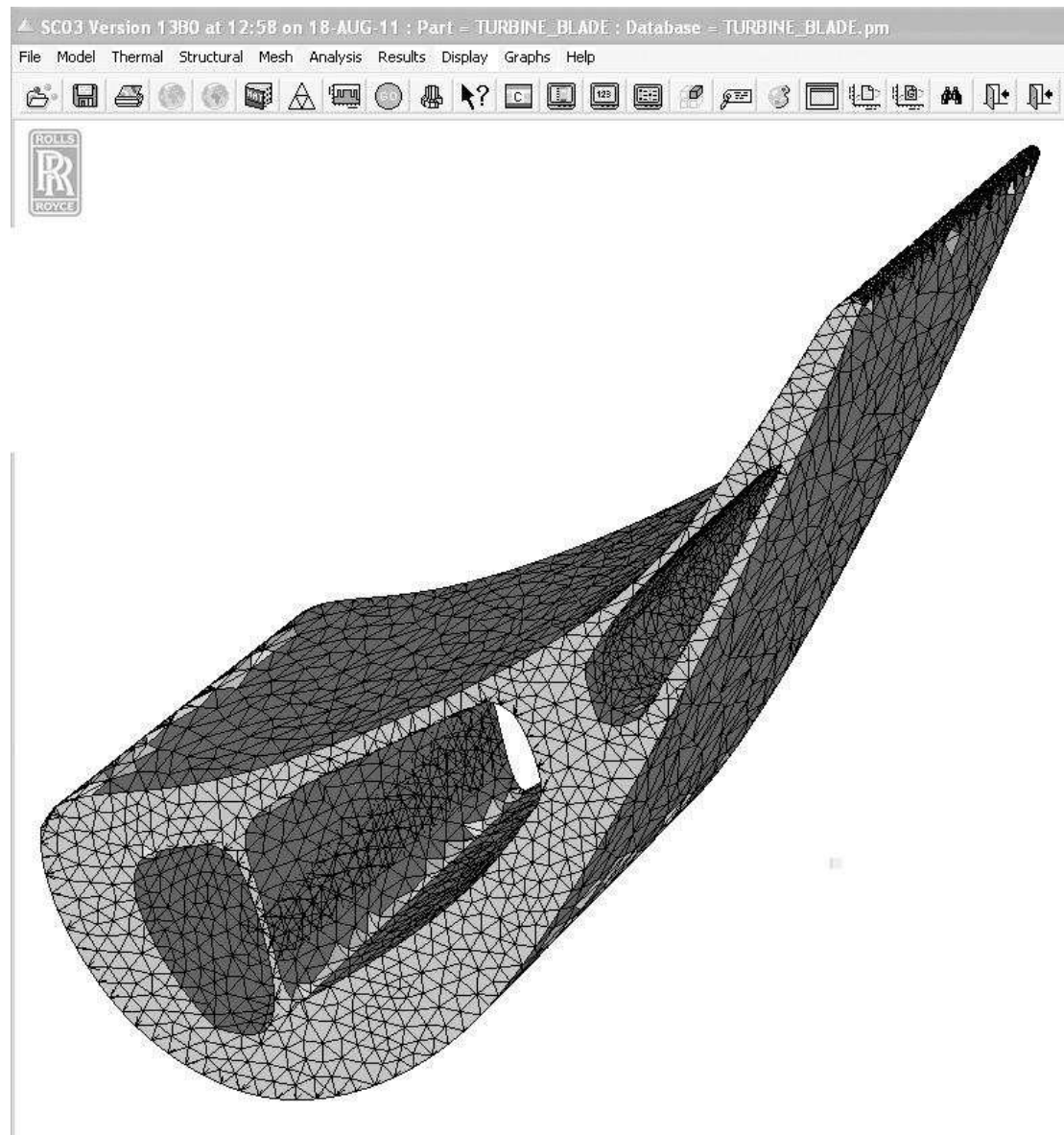


FIGURE 6.7: Blade model meshing

### 6.3 Statistics of modes of failure

The automated design process described above could fail in several different ways, which hinders design automation and reduces the possibility of finding an optimum design near the edge of a feasible design space. The MATLAB script (Appendix A.8.11), which automates the design and analysis process, reports an error code if the process is terminated prematurely.

A sampling plan of 1000 designs is generated using Latin hypercube method, as described in Section 2.4. These designs are put through the design and analysis automation process. Among these 1000 design candidates, 611 fail in the NX model generation process;

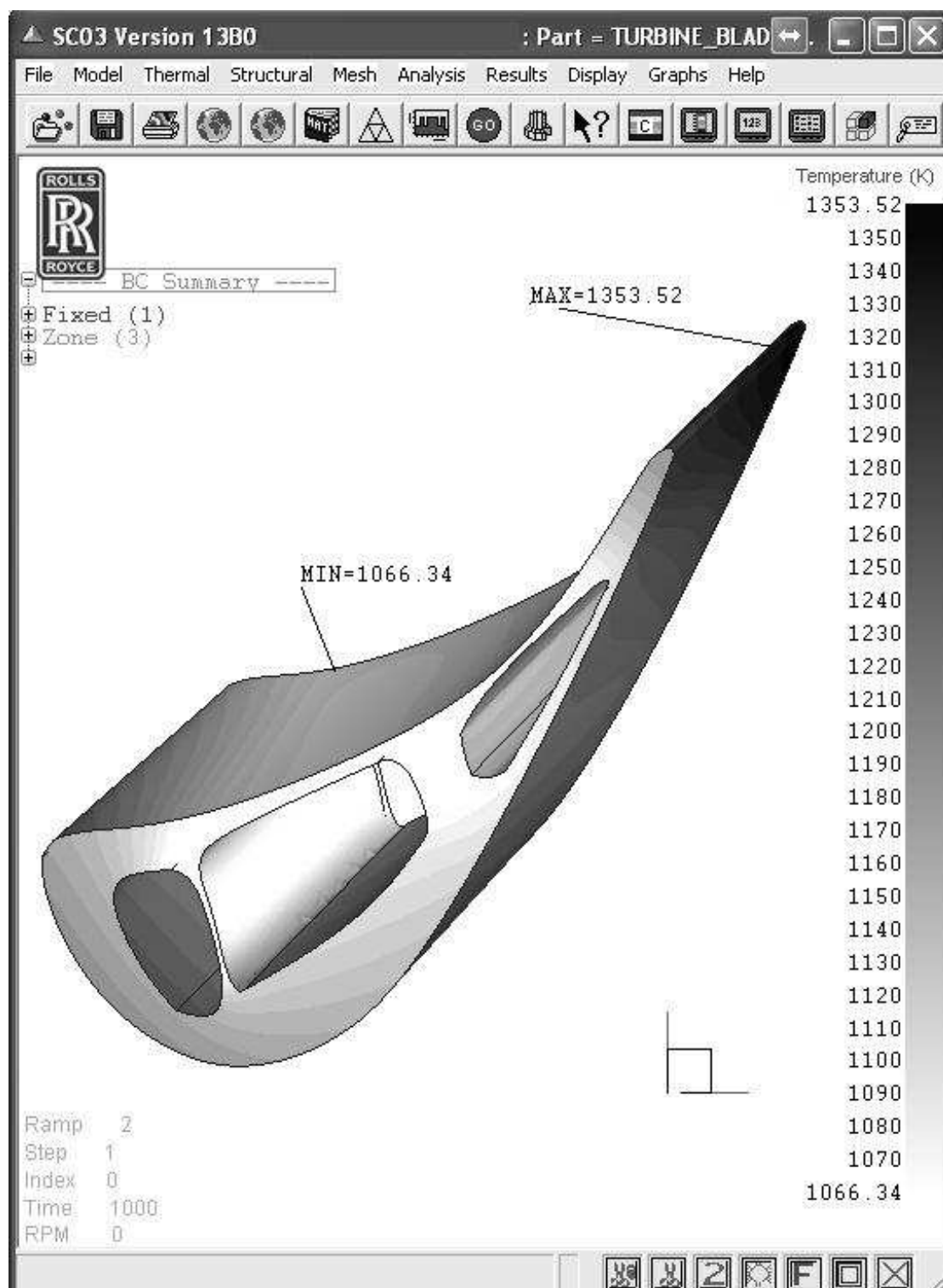


FIGURE 6.8: Blade temperature contours

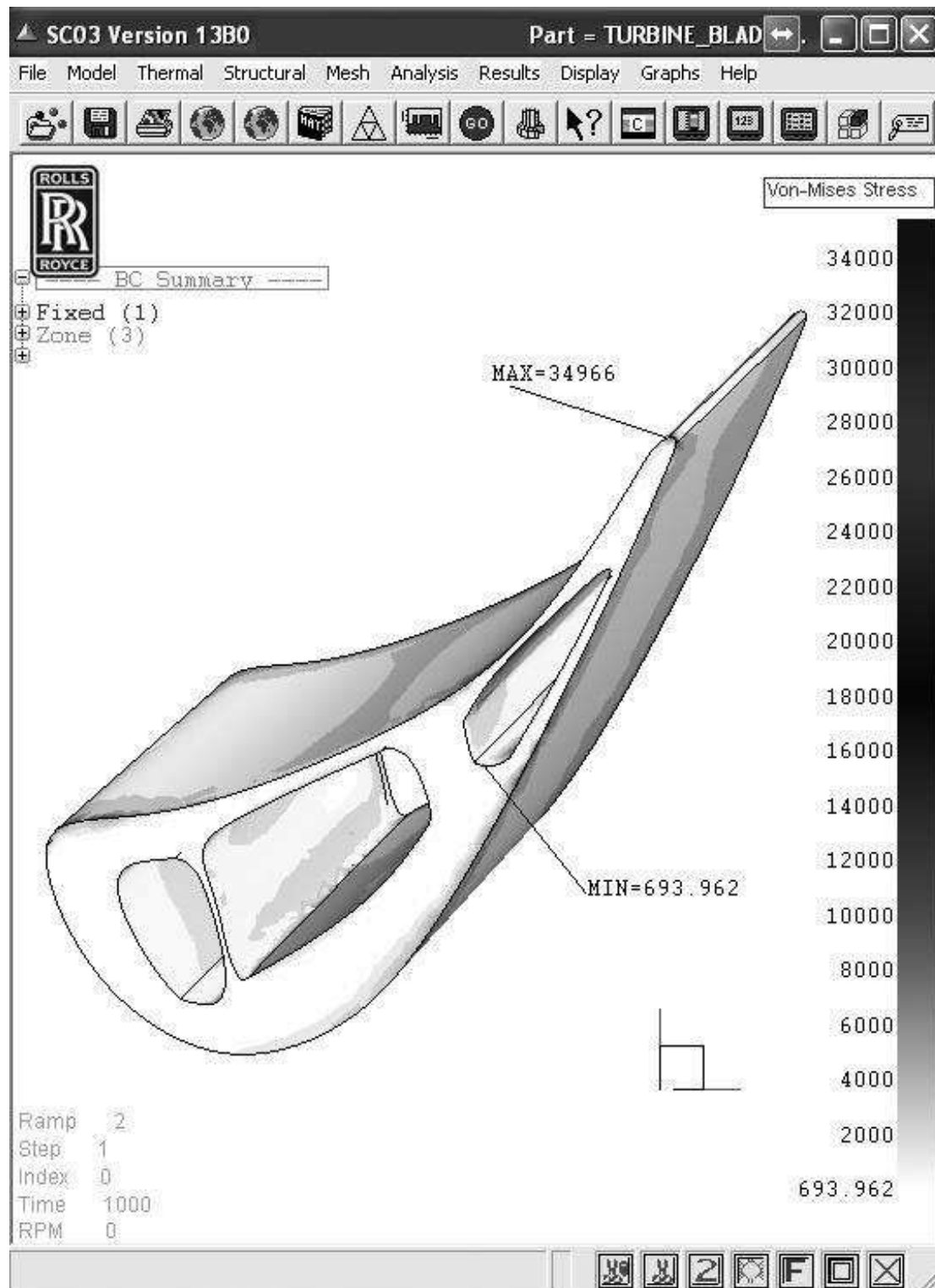


FIGURE 6.9: Von-Mises stress contour map

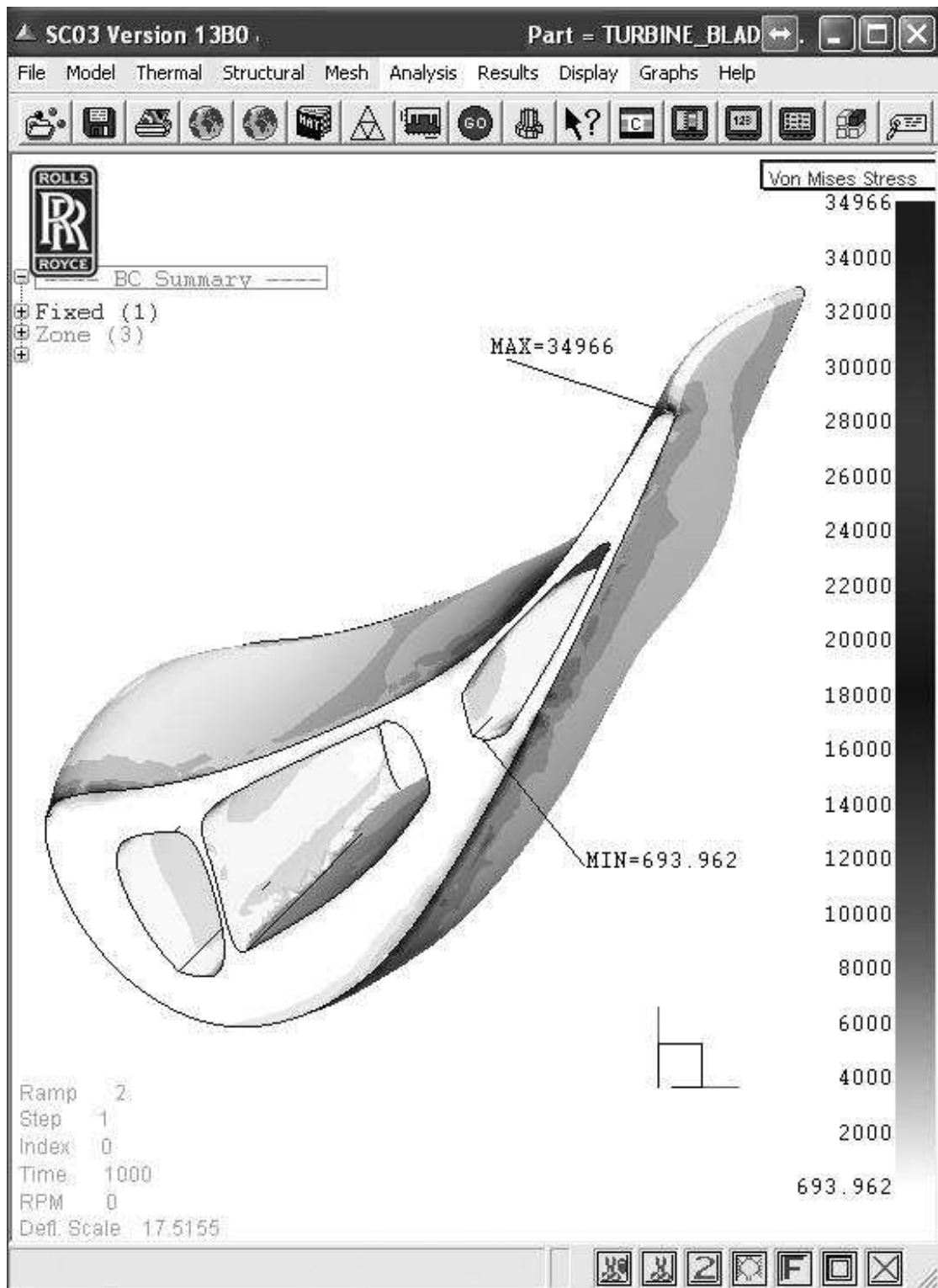


FIGURE 6.10: Von-Mises stress contour map with predicted blade deflection displayed



Error flag	Error	Count
1	NX Part generation failure	611
2	IGES generation failure	0
3	SC03 database generation failure	6
4	Result generation failure	34
0	Design process success	349
		Total:1000

TABLE 6.2: Error type and statistics

none fail during the IGES file generation process; 6 fail in the CADfix process; and 34 fail in SC03 (meshing and analysis). 349 geometries were successfully generated, meshed and analysed. It is observed that the majority failed in the NX geometry generation process. This is probably due to the difficulty of setting up an accurate range for cooling hole positions. The designs with cooling holes intersecting with blade boundaries will fail in the geometry generation process. It is also observed that a significant number of geometries fail in SC03 meshing and analysis process. This is probably due to some designs lead to sharp edges at cross-section of cooling holes, and at cross-section of cooling hole and blade boundaries. The sharp edges could often cause difficulty in meshing.

## 6.4 Knowledge base setup

### 6.4.1 Data collection

As shown in Figure 6.11, different types of data are generated and collected in the geometry generation, translation and analysis process. The 1000 design candidates are put through the process. If the design successfully generates a geometry, the minimum wall thickness of the blade is recorded. If the design is successfully meshed, the largest distortion of the mesh is recorded. After the design passes stress analysis, the maximum temperature and stress are recorded. If the design fails at any stage of the process, an error flag is generated as described in Section 6.3. This error flag is recorded as well.

The maximum temperature recorded for the 1000 geometries is 1406 K and the minimum temperature is 1145.2 K. The maximum von Mises stress recorded for the 1000 geometries is  $2.8 \times 10^5 \text{ N/m}^2$  and the minimum  $2.7 \times 10^4 \text{ N/m}^2$ . The maximum minimum-wall-thickness is 0.4817 millimetre and the minimum 0, which indicates interfering design features. The maximum mesh distortion recorded for the 1000 geometries is 72.39 and the minimum 4.3029. Mesh distortion is a dimensionless number defined by ratio of the radius of the circumscribed sphere to that of the inscribed sphere of a mesh element, and consequently quantifies the extent of distortion. By definition it can be deduced that for a 2 dimensional mesh the minimum possible value of mesh distortion is 2, for an undistorted, equilateral triangle mesh element.

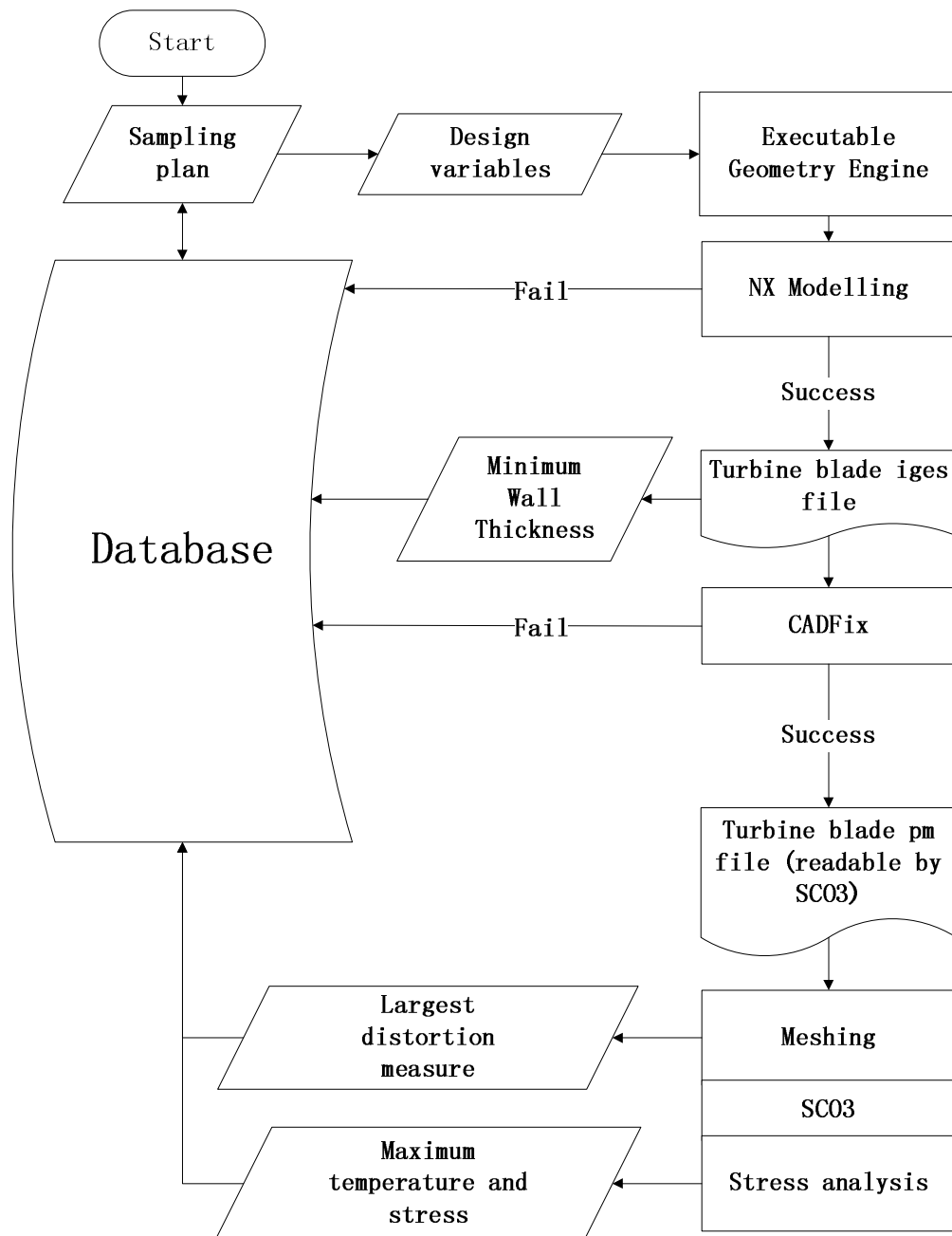


FIGURE 6.11: Data collection process

Design No.	Error flag	Temperature	Von-Mises Stress	Distortion	Minimum wall thickness
1	1	NaN	NaN	NaN	0
2	1	NaN	NaN	NaN	0
3	1	NaN	NaN	NaN	0
4	4	NaN	NaN	NaN	0
5	0	1249.8	34874	10.2613	0.720145
6	0	1390.3	45637	14.5982	0.835647
7	1	NaN	NaN	NaN	0
...	...	...	...	...	...
998	1	NaN	NaN	NaN	0
999	1	NaN	NaN	NaN	0
1000	0	1349.4	45631	8.66375	0

TABLE 6.3: Blade working-condition measurements

	Maximum	Minimum
Temperature (K)	1406	1145.2
Von Mises stress (Pa)	$2.8 \times 10^5$	$2.7 \times 10^4$
Minimum-wall-thickness (millimetre)	0.4817	0
Mesh distortion	72.39	4.3029

TABLE 6.4: Measurement extremes

#### 6.4.2 Geometry quality evaluation and penalty setup

The quality of each candidate geometry is evaluated by measuring its maximum temperature, maximum von Mises stress, maximum meshing distortion and minimum wall thickness.

For infeasible geometries some of whose data are unavailable, the maximum value from that catalogue is used as a substitute for the purpose of geometry quality evaluation and penalty setup. For example, when a design successfully creates a geometry but fails to be processed by SC03, its temperature and stress information is unavailable. The maximum values of these measurements, 1406 K and  $2.8 \times 10^5$  Pa respectively (Table 6.4) are used as substitutes.

After the data are collected, a penalty function is postulated. Following the ideas in Chapter 5, it is desired that less feasible geometries be allocated larger penalty values. The more infeasible a geometry is, the larger a penalty value it should be allocated. Large peak temperature, stress and maximum distortion naturally indicate a unfavourable design. However a large wall thickness is favourable for manufacturing consideration. The following formula is used to convert the original wall thickness data to thickness penalty,  $p_t$ .

$$p_t = 1 - \text{minimum wall distance}, p_t \in [0, 1] \quad (6.9)$$

When a design fails to create a geometry in NX, its minimum-wall-thickness could not be determined. The maximum possible thickness penalty of  $p_t = 1$  is used as the thickness penalty for these failed geometries .

Given the above modification, a penalty metric is assembled in Table 6.5

Design No.	Temperature	Von-Mises Stress	Distortion	Minimum wall thickness
	Penalty			
1	1406	$2.8 \times 10^5$	72.398	1
2	1406	$2.8 \times 10^5$	72.398	1
3	1406	$2.8 \times 10^5$	72.398	1
4	1406	$2.8 \times 10^5$	72.398	1
5	1249.8	34874	10.2613	0.720145
6	1390.3	45637	14.5982	0.835647
7	1406	$2.8 \times 10^5$	72.398	1
...	...	...	...	...
998	1406	$2.8 \times 10^5$	72.398	1
999	1406	$2.8 \times 10^5$	72.398	1
1000	1349.4	45631	8.66375	1

TABLE 6.5: Penalty metric

The average and standard deviation of each column of Table 6.5 is listed in Table 6.6.

	Average	Standard deviation
Temperature (K)	1365.17	74.2
Von Mises stress (Pa)	$1.98 \times 10^5$	$1.14 \times 10^5$
Mesh distortion	52.33	28.23
Minimum-wall-thickness	0.96	0.09

TABLE 6.6: Penalty statistics

The data in Table 6.5 are of different magnitudes. To give each column of data equal consideration, the penalty values are normalised by using Equation 5.8.

A normalised penalty table is generated. The sum of the normalised penalty is used as the geometry quality indicator.

### 6.4.3 Surrogate training and tuning

A SVR surrogate is generated in the same way as described in Section 3.2. The penalty sum and the design parameter table for the 1000 designs were used as a database for the SVR surrogate model training.

Training using the whole database of 1000 data proved prohibitively time-consuming. Various sizes of the subsets of the database were used to generate a SVR surrogate, and the results were assessed by measuring a goodness-of-fit statistic,  $R^2$ .  $R^2$  is the square

Design No.	Temperature	Von-Mises Stress	Distortion	Minimum wall thickness	Penalty Sum
	normalised penalty				
1	0.5501	0.7289	0.7107	0.4710	2.4607
2	0.5501	0.7289	0.7107	0.4710	2.4607
3	0.5501	0.7289	0.7107	0.4710	2.4607
4	0.5501	0.7289	0.7107	0.4710	2.4607
5	-1.5546	-1.4277	-1.4902	-2.5253	-6.9978
6	0.3385	-1.3334	-1.3366	-1.2887	-3.6201
7	0.5501	0.7289	0.7107	0.4710	2.4607
...	...	...	...	...	...
998	0.5501	0.7289	0.7107	0.4710	2.4607
999	0.5501	0.7289	0.7107	0.4710	2.4607
1000	-0.2125	-1.3334	-1.5468	0.4710	-2.6218

TABLE 6.7: Normalised penalty table and penalty sum

of the correlation between the response values and the predicted response values. It is defined as the ratio of the sum of squares of the regression (SSR) and the total sum of squares (SST). SSR is defined as

$$\text{SSR} = \sum_i (\hat{y}_i - \bar{y})^2. \quad (6.10)$$

SST is also called the sum of squares about the mean, and is defined as

$$\text{SST} = \sum_i (y_i - \bar{y})^2. \quad (6.11)$$

$R^2$  is expressed as

$$R^2 = \frac{\text{SSR}}{\text{SST}} \quad (6.12)$$

Size of training subset	$R^2$ calculated using the training set	$R^2$ calculated using all data in the database
100	0.7661	0.0819
110	0.6311	0.0715
120	0.9076	0.1271
130	0.9113	0.1332
140	0.9108	0.1417
150	0.9108	0.1505

TABLE 6.8: Goodness of fit of SVR surrogate model trained by using different sizes of training data

It can be observed from Table 6.8 that goodness of fit improves as the number of training data used increases. However as the number of training data increases, the time required for the computation of the SVR surrogate increases exponentially.

## 6.5 Repair with the knowledge base

An SVR surrogate trained by the first 100 data in the database are used to test the validity of the repair model. Feasibility threshold is set to 1.5566, 80% of the surrogate mean of 1.9457. An infeasible design  $\mathbf{x}_o$  (Figure 6.12) is chosen randomly from the remaining 900 design cases, and is subjected to the repair process. The randomly chosen design  $\mathbf{x}_o$  ranks 315<sup>th</sup> in the 1000 designs.  $\mathbf{x}_o$  generates a NX part that has a shape edge which prevents a successful meshing (Figure 6.13).



FIGURE 6.12: Top view of the original design  $\mathbf{x}_o$  in NX

The variable resolution evolutionary strategy described in Section 3.4 is used to optimise the surrogate landscape in order to find a repair alternative for  $\mathbf{x}_o$ . The optimisation code is modified from the code used for the 2D optimisation used in Chapter 4 to cope

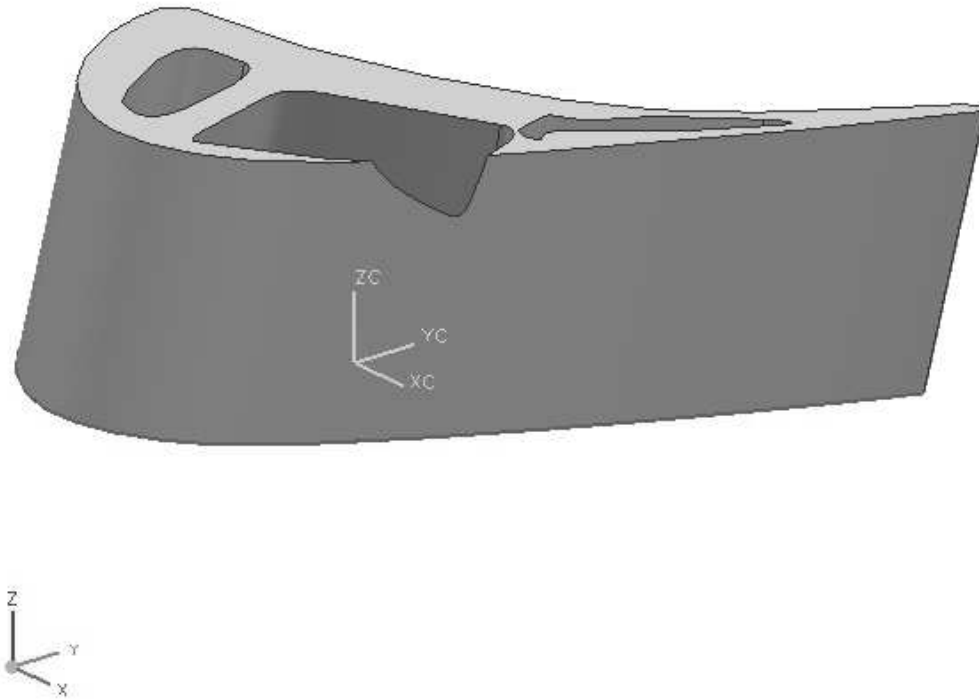


FIGURE 6.13: Trimetric view of the original design  $\mathbf{x}_o$  in NX

with the requirement for a large-scale search, which is necessary for the 16-dimensional design space. The modified code could search a large sampling plan of effectively  $10^7$  in each round of the continuous search in the hyper-circle without exceeding the limit of the random access memory of the author's workstation, which is 3.25GB. Pareto front plots are generated at each stage of the optimisation.

The initial selection process of the design alternative ( $\mathbf{x}_1$ ) is illustrated in Figure 6.14. A repair alternative is found by a global search with a small sampling plan whose size is 12743. Points on the Pareto front is marked by circles. Only one point falls below the feasibility threshold of 1.5566. The point is selected as the initial repair alternative. Its Euclidean distance from the original design  $\mathbf{x}_o$  is 1.101.

After  $\mathbf{x}_1$  is found, optimisation is continued in the hyper-circle centred at the original

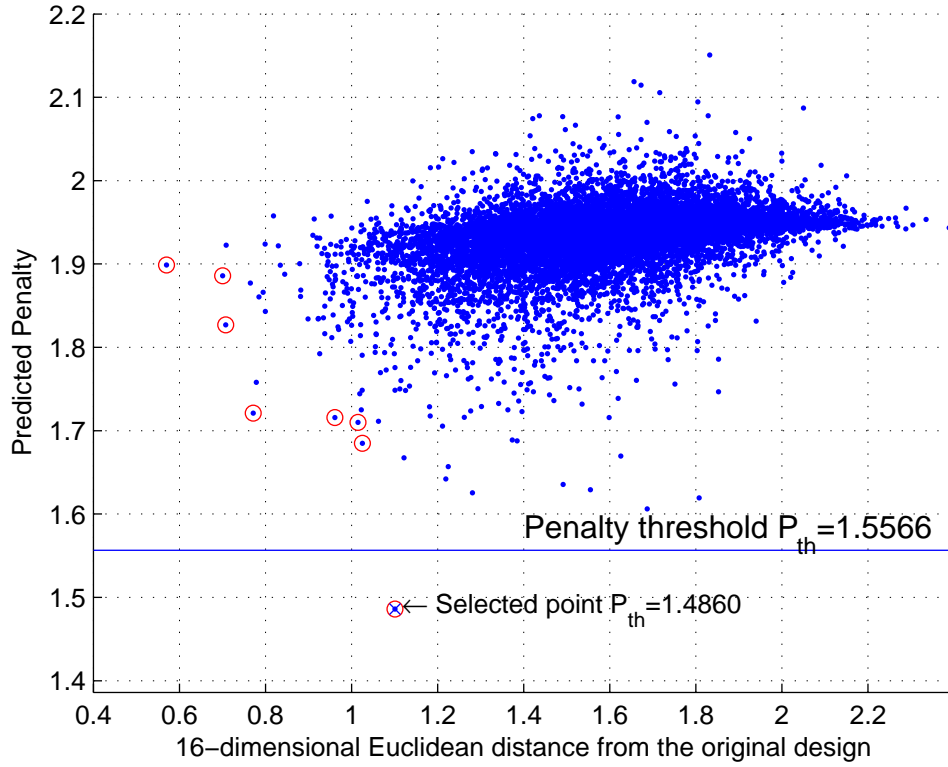


FIGURE 6.14: Initial selection of the repair alternative after a global search

design, with radius equals to current minimum distance of 1.101. Having searched a sampling plan of size  $10^6$ , a better repair alternative  $\mathbf{x}_2$  is found. The searching process is visualised in Figure 6.15: among the points that fall below the feasibility threshold, the one that is closest to the original design is selected as the repair alternative  $\mathbf{x}_2$ .  $\mathbf{x}_2$  is marked by a cross in Figure 6.15. Its Euclidean distance from  $\mathbf{x}_o$  is 0.811, which means that  $\mathbf{x}_2$  is more similar to  $\mathbf{x}_o$  than  $\mathbf{x}_1$  is. The geometry corresponding with the design variable set  $\mathbf{x}_2$  is displayed in Figure 6.16.

After  $\mathbf{x}_2$  is found, subsequent search is conducted in the hyper-circle whose centre is  $\mathbf{x}_o$  and radius is 0.811. Having searched a sampling plan of size 102353, a better repair alternative  $\mathbf{x}_3$  is found. The searching process is visualised in Figure 6.17

$\mathbf{x}_3$  falls below the feasibility threshold of 1.5566. Its Euclidean distance from the  $\mathbf{x}_o$  is 0.5498, which means that  $\mathbf{x}_3$  is more similar to  $\mathbf{x}_o$  than  $\mathbf{x}_2$  is.  $\mathbf{x}_3$  is selected as the repair alternative.

A further search using a sampling plan of effectively  $10^7$  can not find a better repair alternative. This terminates the optimisation, and  $\mathbf{x}_3$  is used as the final repair alternative suggestion  $\mathbf{x}_r$ :

$$\mathbf{x}_r = \mathbf{x}_3. \quad (6.13)$$



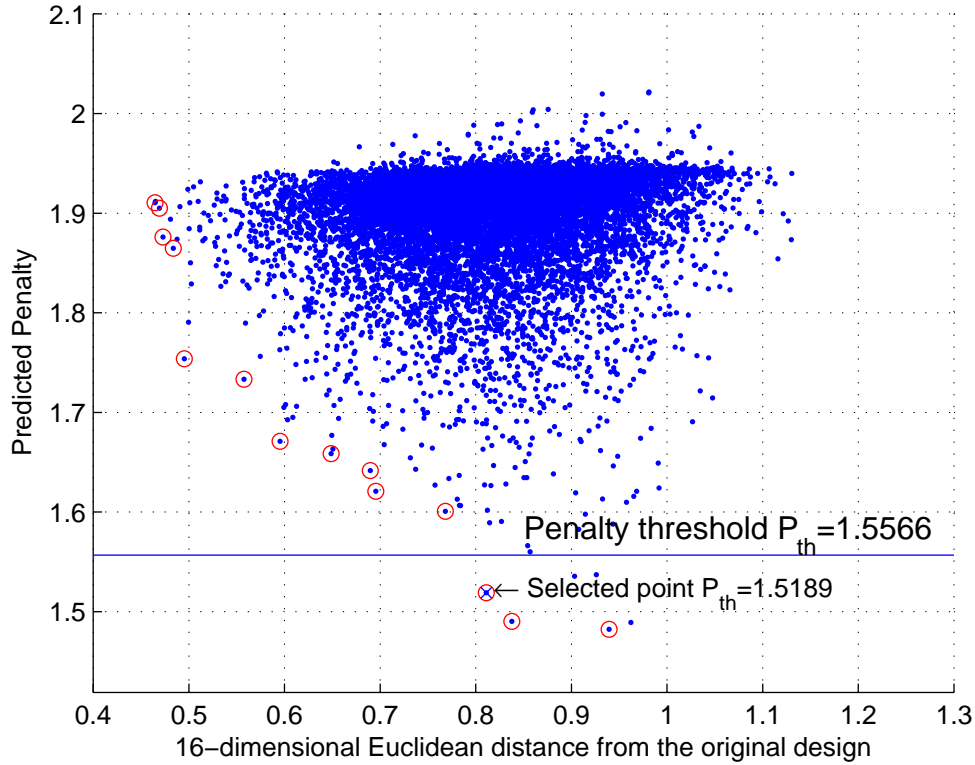
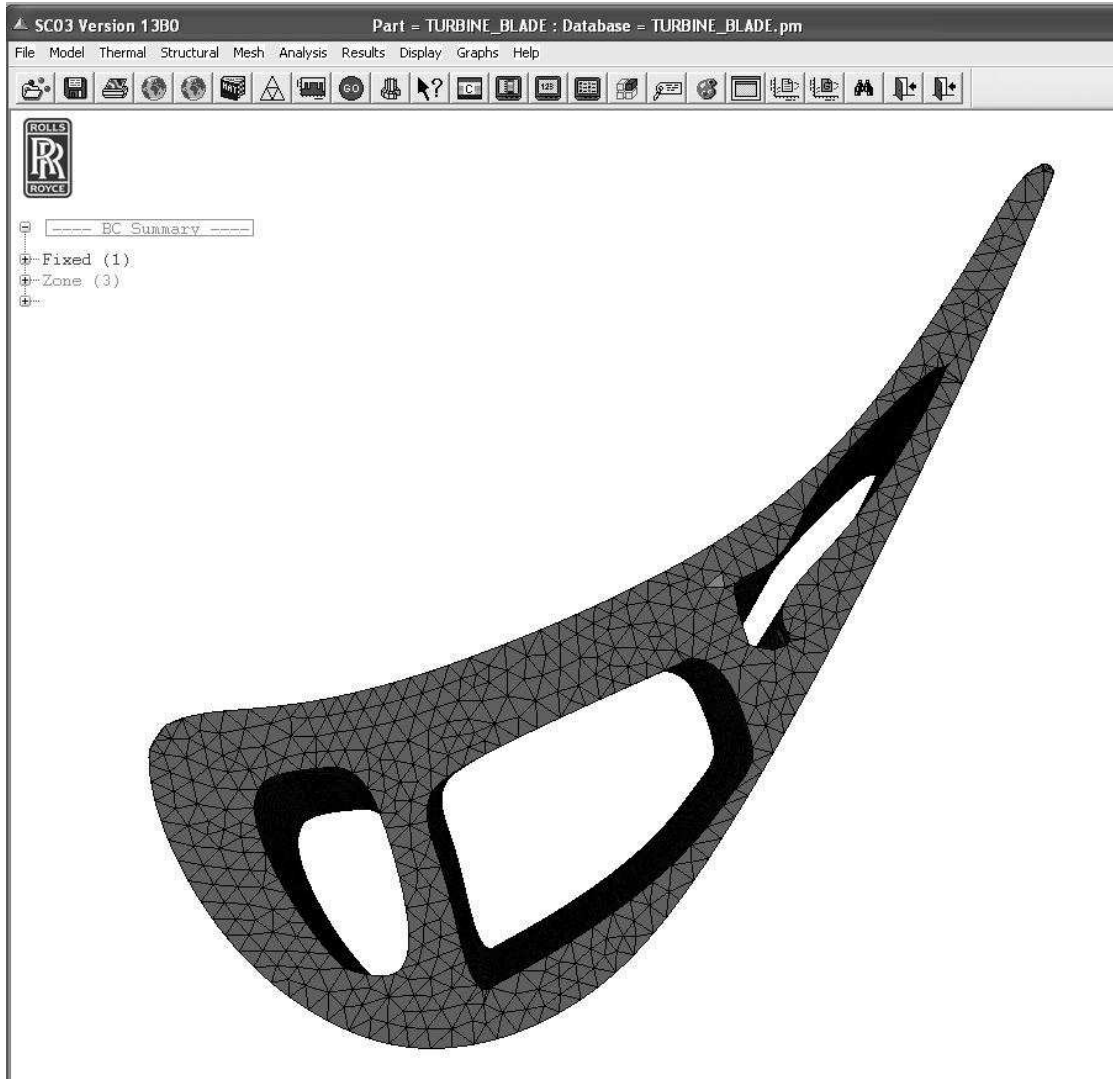


FIGURE 6.15: Selection of the repair alternative in the hyper-circle

The design variables, before and after repair, are presented side by side in Table 6.9 for comparison. By comparing each row of the design variables in Table 6.9, it can be observed that most of the variables in the repaired design variable set is close to the corresponding one in the original design variable set. considering the range of each variable is  $[0, 1]$ . Examining the penultimate row, one can observe that the Euclidean distance from the original design  $\mathbf{x}_o$  gradually reduced, which means the repair alternative become closer to the initial design intent. While at the same time, the last row of Table 6.9 shows that the predicted penalty stays below the preset feasibility threshold.

Comparing Figure 6.12 ( $\mathbf{x}_o$ ) and Figure 6.18 ( $\mathbf{x}_r$ ), it can be observed intuitively that the two designs are still similar. However the sharp edge in Figure 6.12 is gently removed by slightly moving the middle cooling hole toward the inside of the blade. The removal of the shape edge fixes the meshing problem. With a even smaller predicted penalty, Figure 6.16 ( $\mathbf{x}_2$ ) moves the cooling hole further into the blade. This makes sense because in Section 6.4.2, it has been dictated that a larger wall thickness is a favourable feature in manufacturing (Equation 6.9), and the penalty would reduce when the blade thickness is larger.

The selection of feasibility threshold can affect the optimisation as well. Setting a stringent (small) threshold could theoretically produce more feasible designs from physics-

FIGURE 6.16: Repaired blade model  $\mathbf{x}_2$  displayed and meshed in SC03

and engineering knowledge-based point of view. However it could stretch the repair alternative further from the original design. If the threshold is too small, the optimisation may fail. As discussed earlier, if computing budget allows,  $\mathbf{x}_r$  could be generated using multiple  $p_{th}$ , and a designer could view these repair suggestions and select the one that most suit the context of design requirement.

A final test of the efficacy of the repair alternative  $\mathbf{x}_r$  is to test it with the automated turbine blade design and analysis framework described in Section 6.2. Figure 6.19 shows that the suggested design alternative not only successfully generates a geometry, but also makes the meshing and analysis process possible. Similarly,  $\mathbf{x}_2$  also makes the meshing and analysis possible (Figure 6.20). These results show that the proposed repair method is an effective one for the turbine blade design and automation framework.

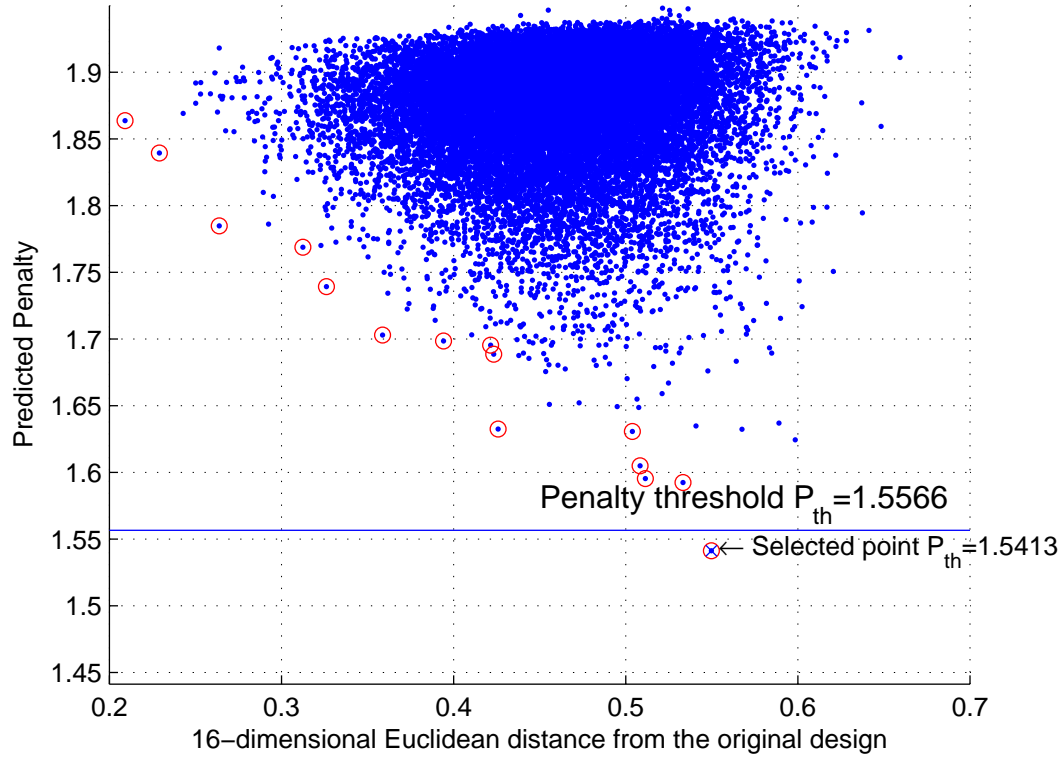


FIGURE 6.17: Selection of the repair alternative in the hyper-circle in subsequent optimisation

$\mathbf{x}$	$\mathbf{x}_o$	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_r$
$x_1$	0.7565	0.7142	0.8600	0.5595
$x_2$	0.3545	0.5548	0.3363	0.4834
$x_3$	0.0195	0.3033	0.1393	0.2059
$x_4$	0.7545	0.3714	1.0975	0.7176
$x_5$	0.3025	0.5818	0.6651	0.4485
$x_6$	0.2995	0.1672	0.2541	0.2312
$x_7$	0.4625	0.2989	0.5625	0.4834
$x_8$	0.5315	0.3542	0.8177	0.6400
$x_9$	0.7645	0.5006	0.8109	0.6162
$x_{10}$	0.8405	0.4233	0.7729	0.6763
$x_{11}$	0.0095	0.5719	0.1863	0.2008
$x_{12}$	0.6625	0.9520	0.8185	0.8133
$x_{13}$	0.8685	0.9556	0.5041	0.7048
$x_{14}$	0.3115	0.3467	0.4766	0.3617
$x_{15}$	0.0895	0.4457	0.2285	0.2353
$x_{16}$	0.4905	0.5869	0.2724	0.3737
Euclidean distance from the original design		1.1010	0.8109	0.5498
Predicted Penalty		1.486	1.5189	1.5413

TABLE 6.9: Original design variables and repair alternatives

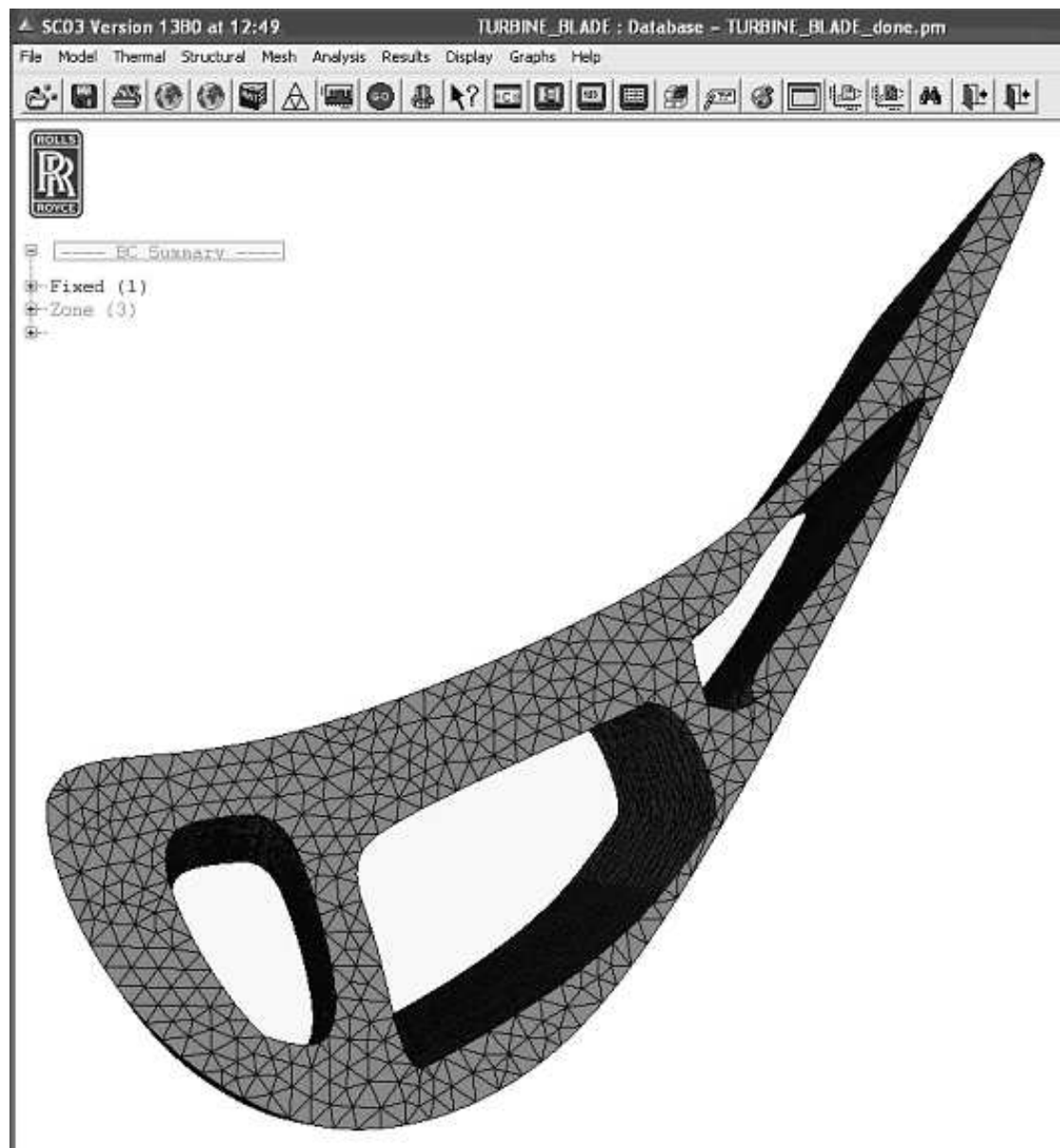


FIGURE 6.18: Suggested repair alternative  $\mathbf{x}_7$  (same as  $\mathbf{x}_3$ ) displayed and successfully meshed in SC03, with the predicted penalty equaling to 1.5413

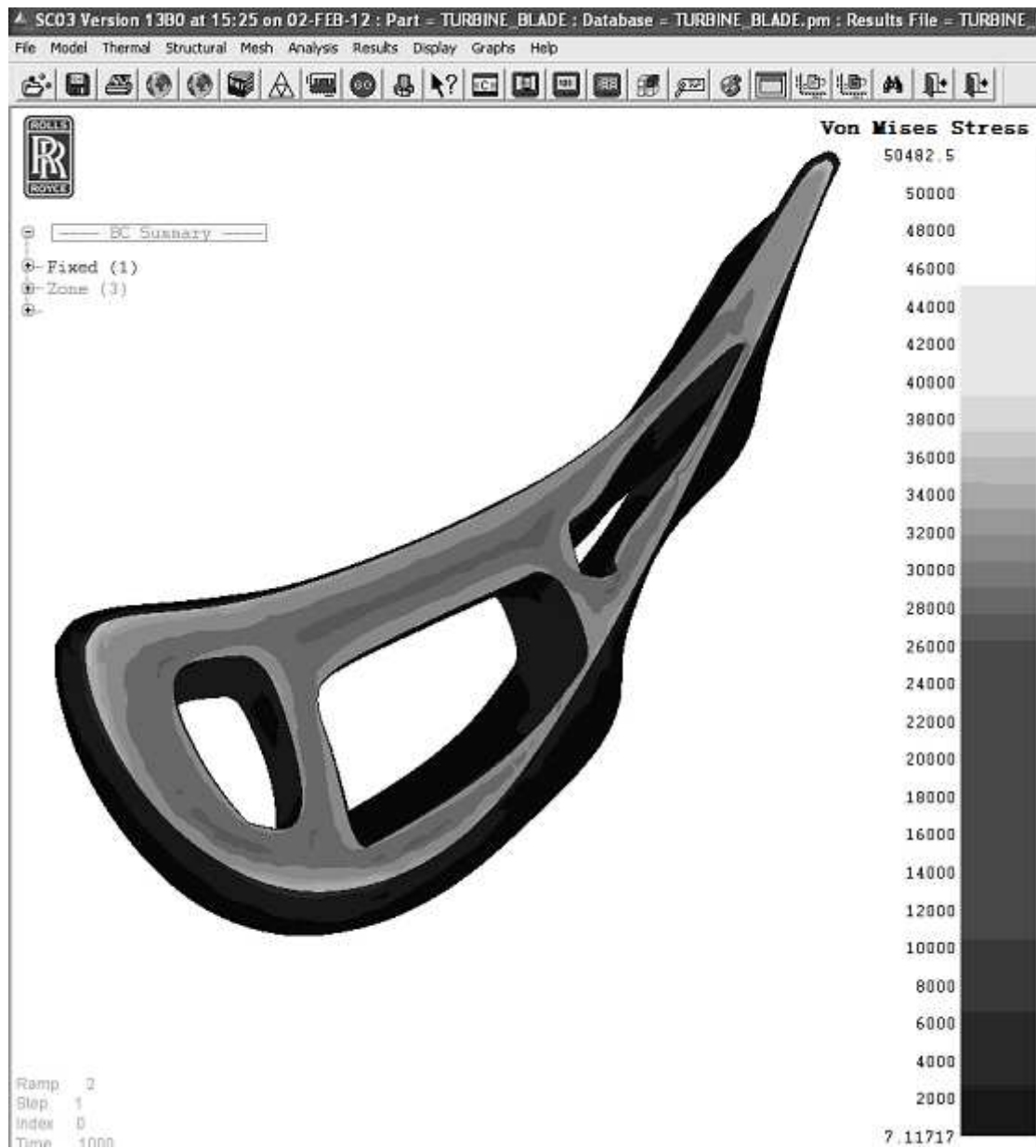
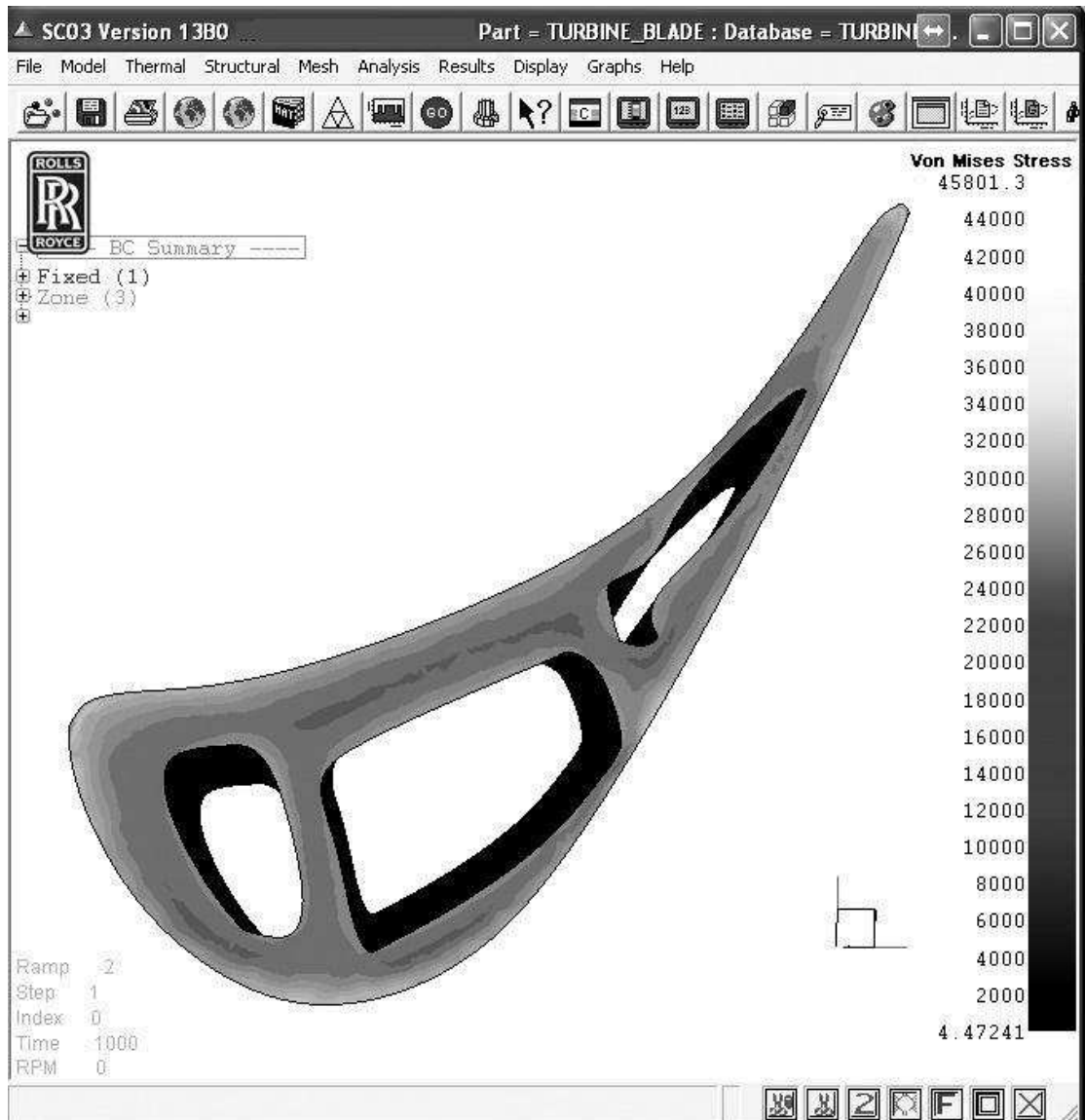


FIGURE 6.19: Final repair alternative  $x_r$  analysed in SC03, showing deflected shape under constraints and thermal stress

FIGURE 6.20: Repaired blade model  $x_2$  analysed in SC03, showing stress contours only



## Chapter 7

# Conclusions

### 7.1 Summary

In an ideal optimisation procedure an effective geometry engine should generate wide ranges of designs without jeopardising geometry robustness. Building a parametric geometry proves a difficult task in reality because there is no universally applicable parameterisation scheme and it is often necessary to compromise between the conflicting goals of geometry robustness and flexibility. This thesis has demonstrated a new approach that tackles this problem by capturing and modelling engineering knowledge, and deploying it automatically in the search for a SPRA, while keeping the original design intent.

In Chapter 1, the background information of computer-aided-design and optimisation was described to put the research objective in perspective. The difficult tradeoff between parameterisation flexibility and robustness was described. Also it was mentioned that the knowledge on feasibility of geometries is seldom collected and reused. The motivation of the thesis, as mentioned in Section 1.1, was to address the problem by developing a physics- and engineering knowledge-based geometry repair system. Section 1.2 provided a clear road map for the thesis.

To understand the background of the research, a wide range of topics in research literature was reviewed in Chapter 2. In Section 2.1, modern computer-aided-design practice in the field of aerospace engineering was reviewed. Two areas that have direct link to this research, i.e., shape parameterization and CFD were reviewed in detail. Mainstream Optimisation techniques were reviewed in section 2.2. Having understood the principle, applicability, advantage and drawback of each technique, the author later developed an ES based optimisation in Chapter 3 for the search of SPRA. In Section 2.3, surrogate modelling techniques were individually examined. Surrogate modelling has wide applications in design optimisation as a means to reduce computing cost. Having



mastered the surrogate modelling techniques, the author demonstrated how to use them as instruments for modelling the feasibility knowledge in later chapters. The author found very limited literature that directly deals with incorporating prior knowledge into surrogate models or geometry repair, which left space for innovation. In summary, by writing this chapter, the author has gained deeper knowledge and a more comprehensive understanding of the research background.

Chapter 3 outlined the proposed knowledge-based geometry repair system, which was first tested in Chapter 4 with a simplified 2D intake geometry design case. Three pieces of engineering knowledge was transformed into forms that can be easily incorporated into the repair system's knowledge base, and Chapter 5 added CFD simulation results to the knowledge base, thus provides a more complete consideration on the feasibility of the intake geometry. The repair system proved a success in generating feasible repair alternatives based on the original, infeasible design, whilst keeping the original design intent to its maximum.

In Chapter 6 the repair system was applied to a 3D turbine blade design case. In this case, an automated turbine blade design and analysis workflow was generated. The feasibility information was collected during the running of the workflow. A repair system was build up based on the same principle described in Chapter 3 and was applied to fixing infeasible geometries. The repair proved a success in Section 6.5.

## 7.2 Contributions

Firstly, the proposed repair system has proved that it can reduce the engineer's workload and increase the design efficiency, flexibility and robustness when it is applied on real scale engineering design problems. The system has provided the ability to repair geometries through an interactive process, where the analyst can readily inspect geometries at various constraint violation levels. Also the system is able to find the nearest feasible alternative as part of an automated search process using a feasibility threshold value. Using the feasible alternative in an optimisation process provides the possibility to make the optimisation cycle continuous and identify potentially optimal design near the boundary of a feasible region, whilst keeping the original design intent as much as possible.

The repair system is also an data fusion tool because the statistics model which lies in the core of the repair system can be trained in a variety of ways using physics- and engineering knowledge, simulation results and individual assessment. One of the contributions of this work is that it proves the possibility of using preset metrics and CFD simulation results for performance and geometric feasibility. This eliminates the tedious process of using manual input to set up a knowledge surrogate.

Thirdly, as the curse of dimensionality demands exponentially increasing numbers of observations as the number of design variables increases, a training process that requires intensive human interaction as per the process outlined in [Sóbester and Keane (2006)] becomes less feasible. A contribution of this thesis is that it provides evidence that a SVR model can be used as the basis of an effective knowledge representation for a high dimensional design space.

Although some ability of dealing with knowledge-based analytical constraints is incorporated into serious optimisation codes, the thesis proposes a specialised way of dealing with constraints in a manner particularly suited to geometry-based design searches. Posing the problem in this way, as mentioned above, allows the analyst to select an acceptable degree of constraint violation in an intuitive, interactive way (Section 4.6), with the same system also functioning as a repair mechanism if deployed in a fully automated setting.

In summary, the repair system can be an invaluable tool for assisting design and optimisation practise in industry.

## **7.3 Recommendations**

### **7.3.1 Recommended application of the work**

The repair system could be used in the conceptual design phase of a product life cycle. The system can be attached to an existing automated optimisation framework to improve its parameterisation flexibility and geometry robustness. The application of the system can lead to a better chance of discovering novel, optimal solution that may not be found using the traditional optimisation approach.

It can also be deployed as a visualisation tool that aids the decision making process. Using the system, the chief engineer can view a set of repair alternatives that range from more feasible designs to risky designs that has the potential to improve existing designs.

### **7.3.2 Recommended future research**

In this section a few pointers to future research are presented.

#### **7.3.2.1 Identifying potential critical designs automatically**

Recently the possibility of identifying potential critical design from SVR process has been examined. The idea has been tested on the traditional quadratic programming support vector regression based on my intake design test model. According to the SVR

theory, the significance of the design is related to the corresponding  $\alpha^\pm$ . For a certain design, if its  $\alpha^\pm = 0$ , this design is not a support vector, and it is effectively ignored in the SVR prediction. If the absolute value of the  $\alpha^\pm$  is large compared to the others for a certain design, this design is a more significant support vector than the others. Thus, an investigation of the distribution of  $\alpha^\pm$  values has been done. As shown in Figure 7.1, for the test case in which 100 samples are used, 57 (57%) of the  $\alpha^\pm$  values lie at the two end of the graph. This amounts to the majority of the designs and is of little help as long as the anticipation here is to identify a few critical designs, which might be helpful in making design decisions. Breaking the histogram graph into more bins Figure 7.2, it is clear that only a small fraction of the design are not support vectors with  $\alpha^\pm = 0$ .

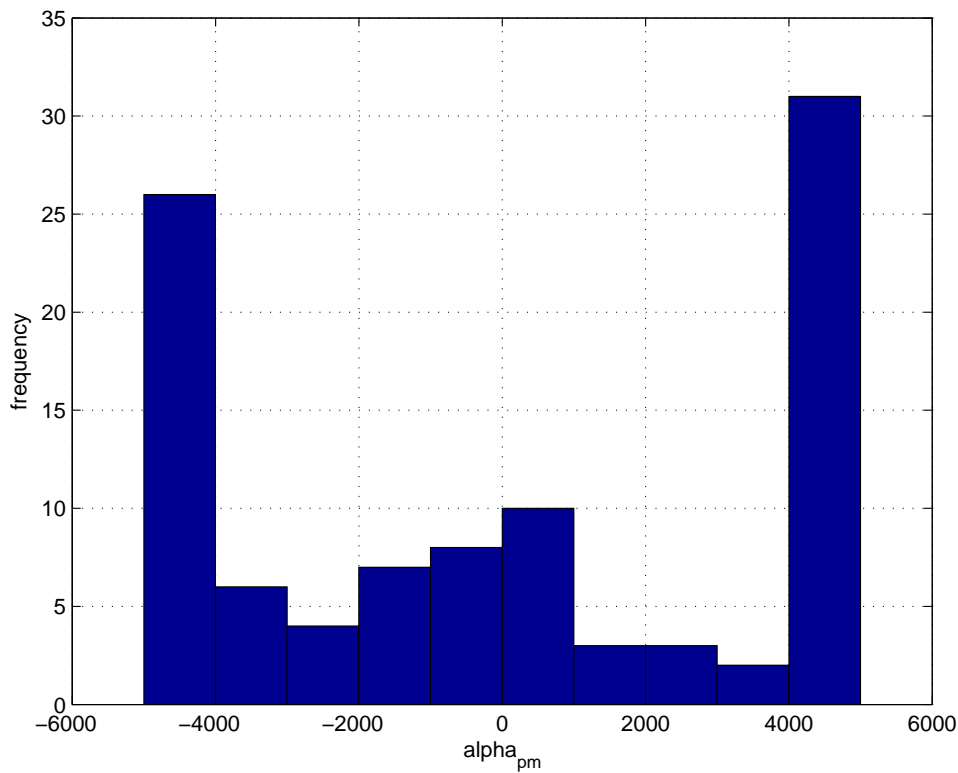
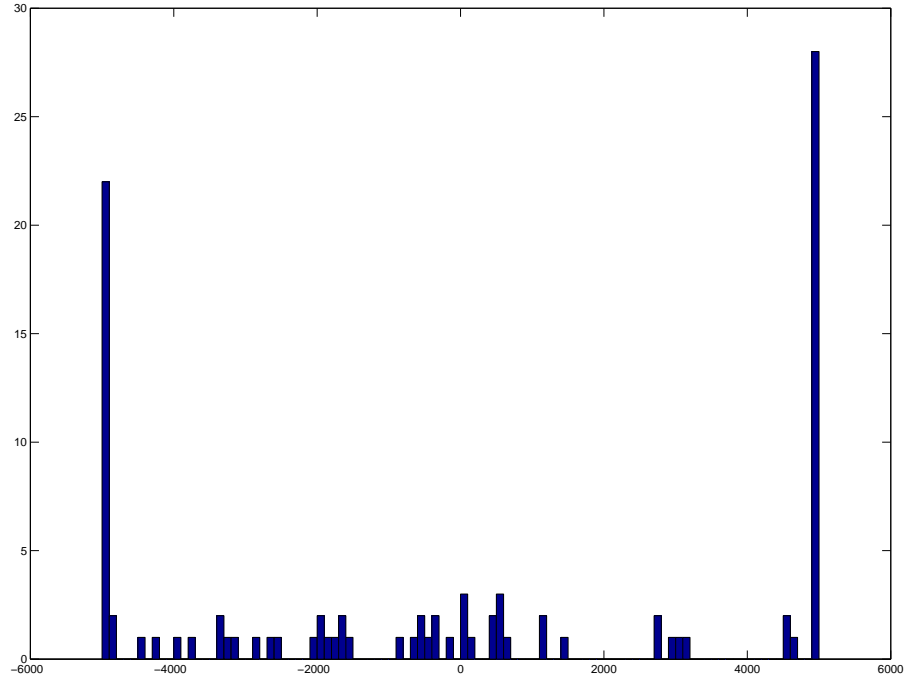


FIGURE 7.1:  $\alpha_{pm}$  histogram (10 bins)

The excess of number of support vectors is possibly due to the sparsity of the sampling points. Thus, most sampling points are regarded as important support vector features. In [Lauer and Bloch (2008)], it is hinted that a linear programming support vector regression (LP-SVR) may significantly reduce the number of support vectors.

FIGURE 7.2:  $\alpha_{pm}$  histogram (100 bins)

### 7.3.2.2 Comparison of other SVR techniques

The reason we use a classic SVR surrogate as the basis for knowledge formation in this work is that the SVR offers sparse representation can give algorithmic and representational advantages in high-dimensional design space. While the classic SVR is used in this work, the author noticed a few variations of the classic SVR being investigated by the wider research community. Two of them are relevance vector machine (RVM) and linear programming support vector regression (LP-SVR). Relevance vector machine (RVM) uses Bayesian inference to obtain parsimonious solutions for regression. The RVM has an identical functional form to the support vector machine, but provides probabilistic classification [Tipping (2001)]. It may offer benefits of probabilistic predictions and exceptional degree of sparsity, which is useful in high-dimensional design space. Investigations of RVM are found in the fields of pattern recognition [Agarwal and Triggs (2006), He et al. (2009)] and neural networks [Yuan et al. (2008)], but no application of using RVM for high-dimensional engineering knowledge modelling has been found. Similarly LP-SVR offer the possibility of further reducing the number of support vectors, and potentially identify critical design in an automatic fashion. It would be interesting to investigate the potentials of these two variations of SVR.

### 7.3.3 Recommendations on research methodology

During the writing of the thesis, the author reflected on the gains and pains during his research. The author highly recommends “write-as-you-work” method. Writing should be as concurrent as possible with other research activities, such as the literature review and computer experiments. Writing is the opportunity to organise ideas, record experimental results and reflect upon these intellectual outcomes. New ideas and experiments result could go missing if they are not collected in time. Write-as-you-work approach greatly reduces the risk of losing work. Admittedly this method could reduce the short-term efficiency. But it is a time saver for the overall research. Never leave the writing all to the end.

The author found that although comprehensive literature review a key to the success of research, one should not try to cover and summarise everything, which would be endless and unproductive. By searching out the literature one should focus on where his own work will contribute to. An engagement in dialogue with what has been written and what is to be written could be inspirational. Look in the reference section of key books and articles of the research can be very helpful in finding useful references. Google scholar is very useful in finding full text papers.

The author felt that it is very important to follow advice from supervisors. At the same time, it is equally important to take the research initiative. This could spark interest and improve motivation. The author also found transfer viva and conference precious opportunities to get constructive criticism from peer students and researchers other than supervisors. The author recommends a good preparation for these events and be prepared to implement some advices in the ensuing research.

# Appendix A

## Important Codes Used in the Thesis

### A.1 Geometry modelling

#### A.1.1 model.m

---

```
%This function draws the intake shape model, including
% 1: The aircraft body by calling drawaircraft.m
% 2. The duct, by calling duct.m and ductoffset.m
%
% Input: x - A set of normalised design variables in a single vector.
% For example:[0,0.4,0.5 0.3]
% mode - 0:plot the figure in a axes
%         1:plot the figure in a new window
% Output:
% A figure showing the geometry corresponding to the design variable
%
% Example: model([0,0.4,0.5 0.3])
% 12/02/2009
function model(x,mode)
if nargin < 2, mode=1; end

% global xmin xmax ymax;%Figure window boundary
xmin=-100; xmax=500; ymax=350;
if mode==0
    drawaircraft_in_axes(xmin,xmax,ymax);
else
    drawaircraft(xmin,xmax,ymax,0);
end

y=0.8*ymax.*x-0.4*ymax;
D=[xmin          0.9*ymax+y(1)*0.5;...
xmin+35          0.9*ymax+y(1)*0.5;...      %D1
0*xmax           0.9*ymax+y(2);...          %D2
0.2*xmax         0.6*ymax+y(3);...          %D3
0.4*xmax         0.5*ymax+y(4);...          %D4
0.5*xmax         0.375*ymax;...             %D5
```

---

```

0.6*xmax      0.375*ymax;];

%Spline parameters assignments and spline calculation
degree=3;
number_of_points=1001;

SPLN=duct(D,degree,number_of_points);
ductoffset(D,SPLN);

```

---

LISTING A.1: model.m

## A.1.2 drawaircraft.m

---

```

%This function draws the aircraft body
%Input: xmin - minimum x coordinate
%       xmax - maximum x coordinate
%       ymax - maximum y coordinate
%       ps - pause time in second (useful when )
% Output:Aircraft boundary in a figure

function drawaircraft(xmin,xmax,ymax,ps)

%Set screen position
scrsz = get(0,'ScreenSize');
figure('Position',[0 scrsz(4)/5+20 scrsz(3)/1.25 scrsz(4)/1.7])
axis([xmin xmax 0 1.2*ymax])
hold on

%The aircraft boundary
B1x=[xmin xmax];
B2x=B1x;
B1y=[0 0.25*ymax];
B2y=[0.75*ymax 0.5*ymax];
line(B1x,B1y,'linewidth',3);
line(B2x,B2y,'linewidth',3);

%The Engine boundary
Ex=[0.6*xmax xmax xmax 0.6*xmax 0.6*xmax];
Ey=[0.25*ymax 0.25*ymax 0.5*ymax 0.5*ymax 0.25*ymax];
plot(Ex,Ey,'k','linewidth',2);
text(0.6*xmax,0.37*ymax,' \leftarrow Engine Intake','FontSize',12)

%Draw bulkhead, this bulkhead is pre-calculated and stored in a mat file
%called 'B.mat'. This is actually a stupid idea since this diminish the
%flexibility of drawing the model under different parameters(xmin, xmax, ymax)
load Bulkhead;
szub=10001;
plot(B(1,1:szub),B(2,1:szub),'k','LineWidth',2)

grid on
pause(ps)
%Note:
%1. The bulkhead is pre-calculated and stored in a mat file
%called 'B.mat'. This is actually a stupid idea since this diminish the
%flexibility of drawing the model under different parameters(xmin, xmax, ymax)

```

---

LISTING A.2: drawaircraft.m

### A.1.3 duct.m

---

```

%This function calculates the B-spline
% Input:    DESIGN_VARIABLE -
%           (n by 2 matrix, can be conveniently generated from model.m)
%           degree - Degree of the B-Spline, three is a common used value
%           number_of_points - number of points used to approximate the B-Spline
% Output:    SPLN: The centre B-spline which is approximated by 1000 points.
%
function SPLN=duct(DESIGN_VARIABLE,degree,number_of_points)
hold on

D=DESIGN_VARIABLE;
p=degree;

%knot vector construction
szC=size(D,1);
U=[zeros(1,p),0:1/(szC-p):1,ones(1,p)];
szU=size(U,2);           %Size of knot vector
interval=1/(number_of_points-1);
u=U(1+p):interval:U(szU-p);
szu=size(u,2);           %Return the size of u
N=zeros(szu,szu);        %Define the storage matrix for the basis func.

for i=1:(szU-1)           % 0 degree basis function
    for j=1:szu
        if (u(j)>=U(i)&&u(j)<U(i+1))
            N(i,j)=1;
        else
            N(i,j)=0;
        end
    end
end

for i=1:p+1
    N(szu-i,szu)=1;
end

for i=1:p                 %Degree p basis function
    for j=1:(szU-i-1)
        for k=1:szu
            if U(j+i)-U(j)==0
                add1=0;
            else
                add1=(u(k)-U(j))/(U(j+i)-U(j))*N(j,k);
            end
            if U(j+i+1)-U(j+1)==0
                add2=0;
            else
                add2=(U(j+i+1)-u(k))/(U(j+i+1)-U(j+1))*N(j+1,k);
            end
            N(j,k)=add1+add2;
        end
    end
end
end

```



---

```

SPLN=zeros(2,szu);
for i=1:szC
    for j=1:szu
        SPLN(1,j)=SPLN(1,j)+N(i,j)*D(i,1);
        SPLN(2,j)=SPLN(2,j)+N(i,j)*D(i,2);
    end
end

%Notes:
% Using 1001 points to draw the figure, the figure looks the same,
% but calculation is much faster .

```

---

LISTING A.3: duct.m

### A.1.4 ductoffset.m

---

```

%This function calculates the offset of the B-spline and add the duct shape
%onto the current figure if it exist.
% Input:
%         DESIGN_VARIABLE -
%         (n by 2 matrix, can be conveniently generated from model.m)
%         SPLN: The centre B-spline, which is a 2 by n matrix
%
% Output:
%         Add the duct centre line as well as the offset onto the current figure

function ductoffset(DESIGN_VARIABLE,SPLN)

%Assignments
D=DESIGN_VARIABLE;
plotpolyline='on';

%Generate offset splines
szS=size(SPLN,2);
plot(SPLN(1,1:szS),SPLN(2,1:szS),'--k','LineWidth',1)
diff=zeros(2,szS-1);
for i=1:szS-1
    for j=1:2
        diff(j,i)=SPLN(j,i+1)-SPLN(j,i);
    end
end

for i=1:szS-1
    if diff(2,i)~=0
        t=-diff(1,i)/diff(2,i);
        diff(1,i)=43.75/(1+t^2)^(1/2);
        diff(2,i)=43.75*t/(1+t^2)^(1/2);
        if diff(2,i)<0
            diff(1,i)=-diff(1,i);
            diff(2,i)=-diff(2,i);
        end
    else
        diff(1,i)=43.75;
        diff(2,i)=0;
    end
end
end

```

---

```

%Create Upper offset line
uofs=zeros(2,szS-1);
for i=1:szS-1
    for j=1:2
        uofs(j,i)=SPLN(j,i)+diff(j,i);
    end
end
plot(uofs(1,:),uofs(2,:), 'r', 'linewidth', 2)
%Create Lower offset line
lofs=zeros(2,szS-1);
for i=1:szS-1
    for j=1:2
        lofs(j,i)=SPLN(j,i)-diff(j,i);
    end
end
plot(lofs(1,:),lofs(2,:), 'r', 'linewidth', 2)

%Plot ployline and Spline
if strcmp(plotpolyline, 'on')
    Cx=D(:,1); Cy=D(:,2);
    plot(Cx,Cy, 'o'), grid on
    line(Cx,Cy)
end

%Add the duct centre line onto the current figure
plot(SPLN(1,:), SPLN(2,:), '--r', 'LineWidth', 1)

```

---

LISTING A.4: ductoffset.m

## A.2 Penalty functions

### A.2.1 penalty.m

---

```

function p=penalty(D)
%Weighted Combined penalty
%Input: Design variable D
%Output: combined penalty
p=2*penalty1(D)+penalty2(D)+penalty3(D)/7;

```

---

LISTING A.5: penalty.m

### A.2.2 penalty1.m (abridged)

---

```

function pnty=penalty1(x)
%Input: x - design variable set
%Output: pnty - distance penalty
...
y_distance=lofs(2,1)-0.75*y_max;
if y_distance >= 0
    pnty=y_distance^1.5;
else
    pnty=-100*y_distance;

```

---

---

end

---

LISTING A.6: penalty1.m

### A.2.3 penalty2.m (abridged)

---

```
function pnty=penalty2(x)
%Excessive curvature penalty function
%Input: x - design variable set
%Output:pnty - curvature penalty
...
% Curvature calculaion
DRVT_Augmented=[DRVT;zeros(1,szud)];
SDRVT_Augmented=[SDRVT;zeros(1,szudd)];
Crossp=cross(DRVT_Augmented,SDRVT_Augmented,1);
Cnorm=abs(Crossp(3,:));
DRVT_norm=zeros(1,szud);
for i=1:szud
    DRVT_norm(i)=norm(DRVT(:,i));
    DRVT_norm(i)=DRVT_norm(i)^3;
end

%Curvature
Curvature=Cnorm./DRVT_norm;

%Radius of Circle of Curvature
for i=1:size(Curvature,2)
    if Curvature(i)==0
        Curvature(i)=realmin;
    end
end %Avoid devided by zero
RadiusoCoC=1./Curvature;

%Calcute the penalty
finish_index=size(RadiusoCoC,2);
start_index=ceil(0.1*finish_index);
%We do this because we don't want to include the head of the duct into
%consideration.

pnlt=0;
% Penalty curvature threshold
pcth=43.75;

for i=start_index:finish_index
    if RadiusoCoC(i)<pcth
        pnlt=pnlt+(pcth-RadiusoCoC(i));
    end
end

%Magnify the penalty value so that it is comparable with other penalties
pnlt=pnlt*15;
```

---

LISTING A.7: penalty2.m

### A.2.4 penalty3.m (abridged)

---

```

function pnty=penalty3(x)
%Bulkhead interference penalty function
%Input: x - design variable set
%Output:pnty - interference penalty
...
load Bulkhead.mat
p3=0;
for i=35:-1:-40
    indexB=find(B(1,5000:10001)<=i);
    indexL=find(lofs(1,:)>=i);
    ydiff=lofs(2,indexL(1))-B(2,indexB(1)+4999);
    if ydiff<0
        p=-10*ydiff;
    else
        p=0;
    end
    p3=p3+p;
end
end

```

---

LISTING A.8: penalty3.m

## A.3 Support vector regression and prediction

### A.3.1 SVR regression function.m

This function is adapted from the SVR prediction source code in [Forrester et al. (2008)].

---

```

clear

global X e alpha_pm sigma mu

load X200
load y_200
X=X200(1:200,:);
y=y_200(1:200);

sigma=0.5;
C=100;
nu=0.9;

n=size(X,1);
options = optimset('MaxIter',1e20);

% user defined constants
xi=1e-15;
% build correlation matrix (set theta=30
% arbitrarily without tuning
Psi=zeros(n,n);
for i=1:n
    for j=1:n
        Psi(i,j)=exp(norm(X(i,:)-X(j,:))^2*(-1/sigma^2));
    end
end
end

```

---

```

Psi=Psi+eye(n,n)*1e-10;
% matrix of correlations
Psi=[Psi -Psi;-Psi Psi];
% constraint terms
c=[-y;y];
% lower bound |alpha|>=0
lb=zeros(2*n,1);
% lower bound |alpha|<=0
ub=(C/n)*ones(2*n,1);
% start at alpha=[0;0;...;0]
x0=zeros(2*n,1);
% set sum(alpha^+ + alpha^-)<=C*nu
A = [ones(1,n) +ones(1,n)];
b=C*nu;
% set sum(alpha^+ - alpha^-)=0
Aeq=[-ones(1,n) ones(1,n)];
beq=0;
% run quadprog
alpha=quadprog(Psi,c,A,b,Aeq,beq,lb,ub,x0,options);
% combine alphas into nx1 vector of SVs
alpha_pm=alpha(1:n)-alpha(n+1:2*n);

% find indices of SVs
sv_i=find(abs(alpha_pm)>xi);
num_svs=length(sv_i);
num_svs/n;
% find SVs mid way between 0 and C/n for e and mu calculation
[sv_mid_p,sv_mid_p_i]=min(abs(abs(alpha(1:n))-(C/(2*n)))));
[sv_mid_m,sv_mid_m_i]=min(abs(abs(alpha(n+1:2*n))-(C/(2*n)))));
% calculate e
e=0.5*(y(sv_mid_p_i)-y(sv_mid_m_i)-alpha_pm(sv_i)'*Psi(sv_i,sv_mid_p_i)+alpha_pm(
    sv_i)'*Psi(sv_i,sv_mid_m_i));
% calculate mu
mu=y(sv_mid_p_i)-e*sign(alpha_pm(sv_mid_p_i))-alpha_pm(sv_i)'*Psi(sv_i,sv_mid_p_i
    );

save SVR_RESULT e alpha_pm sigma mu

```

---

LISTING A.9: SVR regression function

### A.3.2 SVR prediction function

---

```

function pred=SVR_Y_Gaussian_pred(x)
%Input: row vector x
%Output: predicted value
global X alpha_pm sigma mu
n=size(X,1);
psi=zeros(n,1);
for i = 1:n
    psi(i)=exp(norm(x-X(i,:))^2*(-1/sigma^2));
end

pred=mu+alpha_pm'*psi;

```

---

LISTING A.10: SVR prediction function.m

## A.4 ES optimisation repair function

[caption=ES optimisation repair function]

---

```
% function evop_repair determines a repair alternative for design D0 based on a
% prediction
% function and a penalty threshold.
%
% Input - X0: original design variable set
%         objhandle: name of the prediction function
%         th: the prescribed penalty threshold value
%
% Output- X_best: suggested repair alternative
%
% Example:
% objhandle='SVR_Gaussian_pred';
% th=1500;
% X0=[0.2, 0.1, 0.5, 0.5];
% X_best=evop_repair(X0,objhandle,th)
%
% Do not forget to load the surrogate parameters
%
% Dong Li 10/06/2010

function X_best=evop_repair(X0,objhandle,th)
% Initial global search
for i=5:2:19
    [SAMPLE_PLAN,sample_plan_size]=sample_plan(i,'rlh');
    RESULT=zeros(sample_plan_size,2); RESULT(:)=Inf;
    for j=1:sample_plan_size % search x1 in the whole design space
        RESULT(j,1)=norm(X0-SAMPLE_PLAN(j,:)); % Calculate the
        distance
        RESULT(j,2)=feval(objhandle,SAMPLE_PLAN(j,:)); % Calculate the penalty
    end

    [dmin,indexbp]=best_point(RESULT,th);
    if dmin<inf
        X1=SAMPLE_PLAN(indexbp,:);
        fprintf('A repair alternative has been found by an initial %d^4 global
search, continue with hyper-circle search\n',i);
        X_best=X1;
        break
    else
        fprintf('No repair alternative can be found with less than 19^4 search
points for th=%d, return X_best=X0 \n',th)
        X_best=X0;
        return
    end
end

% Initial search in the hyper-circle centered at the original design,with radius
==dmin
for i=5:2:21
    [SAMPLE_PLAN,sample_plan_size]=sample_plan(i,'rlh');
    SAMPLE_PLAN=SAMPLE_PLAN*(2*dmin)-dmin;

    for j=1:sample_plan_size %This operation isn't slow.. don't need to change
        to 100..

```

---

```

        SAMPLE_PLAN(j,:)=X0+SAMPLE_PLAN(j,:); %Generate the second sample plan
    end
    RESULT=result(SAMPLE_PLAN,X0,dmin,objhandle);
    [d,indexbp]=best_point(RESULT,th);
    Xc=SAMPLE_PLAN(indexbp,:);
    if d<dmin
        X_best=Xc;
        dmin=d;
        break
    end
end
if X_best==X1
    return
end

% Continuous circle search
Xc=[0 0 0 0];
count=0;
while (~isequal(Xc,X_best))
    X_best=Xc;
    count=count+1;
    for i=5:15
        %Generate a sample plan centered at X0 with side length=2*dmin
        [SAMPLE_PLAN,sample_plan_size]=sample_plan(i,'rlh');
        SAMPLE_PLAN=SAMPLE_PLAN*(2*dmin)-dmin;
        for j=1:sample_plan_size %This operation isn't slow.. don't need to
change to 100..
            SAMPLE_PLAN(j,:)=X0+SAMPLE_PLAN(j,:); %Generate the second sample
plan
        end
        RESULT=result(SAMPLE_PLAN,X0,dmin,objhandle);
        [d,indexbp]=best_point(RESULT,th);
        if d<dmin
            Xc=SAMPLE_PLAN(indexbp,:);
            dmin=norm(X0-Xc);
            break
        end
    end
end
end
end

```

---

LISTING A.11: ES optimisation repair function

## A.5 MATLAB codes as described in Figure 5.1

---

```

%Automated geometry generation, GAMBIT meshing and FLUENT solving for the
%two dimensional duct. Store the total pressure data at the exit of the duct.

%Automatically read the data file from hard drive. Calculate the standard
%deviation of the pressure. The standard deviation is used as a measure
%of the fan face distortion, and is regarded as a metric of the engineering
%feasibility of the duct.

%To switch to any other sample plan, only need to modify line 14 18,22&25.
%Dong Li
%15/05/2010

```

```

% v6 is adapted from v4.3 as a piece of code that can assess an input of four
    variables.
% input directly the parameters of the design
% fluent simulation iteration is reduced to 10

%Housekeeping
% clear
close all

% std_data=zeros(1,1);

sample_nmbr=1000;% assign to a fake number
fprintf('Working on Geometry%d...\n',sample_nmbr);
x=[0.4838      0.3980  0.1064  0.5746];

% global xmin xmax ymax;%Figure window boundary
xmin=-100;ymax=500;ymax=350;
close
draw_duct(xmin,xmax,ymax,0);

y=0.8*ymax.*x-0.4*ymax;
D=[xmin      0.9*ymax+y(1)*0.5;...
xmin+35      0.9*ymax+y(1)*0.5;...      %D1
0*xmax      0.9*ymax+y(2);...      %D2
0.2*xmax      0.6*ymax+y(3);...      %D3
0.4*xmax      0.5*ymax+y(4);...      %D4
0.5*xmax      0.375*ymax;...      %D5
0.6*xmax      0.375*ymax;];

%Spline parameters assignments and spline calculation
degree=3;
number_of_points=1001;

SPLN=duct(D,degree,number_of_points);

%Assignments

plotpolyline='on';

%Generate offset splines
szS=size(SPLN,2);
plot(SPLN(1,1:szS),SPLN(2,1:szS),'--k','LineWidth',1)
diff=zeros(2,szS-1);
for i=1:szS-1
    for j=1:2
        diff(j,i)=SPLN(j,i+1)-SPLN(j,i);
    end
end

for i=1:szS-1
    if diff(2,i)~=0
        t=-diff(1,i)/diff(2,i);
        diff(1,i)=43.75/(1+t^2)^(1/2);
        diff(2,i)=43.75*t/(1+t^2)^(1/2);
        if diff(2,i)<0
            diff(1,i)=-diff(1,i);
            diff(2,i)=-diff(2,i);
        end
    end
end

```



```

        end
    else
        diff(1,i)=43.75;
        diff(2,i)=0;
    end
end

%Create Upper offset line
uofs=zeros(2,szS-1);
for i=1:szS-1
    for j=1:2
        uofs(j,i)=SPLN(j,i)+diff(j,i);
    end
end
while uofs(1,1)<-100.1
    uofs(:,1)=[];
end
plot(uofs(1,:),uofs(2,:), 'r', 'linewidth', 2)
%Create Lower offset line
lofs=zeros(2,szS-1);
for i=1:szS-1
    for j=1:2
        lofs(j,i)=SPLN(j,i)-diff(j,i);
    end
end
while lofs(1,1)<-100.1
    lofs(:,1)=[];
end
plot(lofs(1,:),lofs(2,:), 'r', 'linewidth', 2)
%Plot ployline and Spline
if strcmp(plotpolyline, 'on')
    Cx=D(:,1);Cy=D(:,2);
    plot(Cx,Cy, 'o'), grid on
    line(Cx,Cy)
end

%Add the duct center line onto the current figure
plot(SPLN(1,:),SPLN(2,:), '--r', 'LineWidth', 1)
saveas(gcf, strcat('C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\',
    num2str(sample_nمبر), '.eps'))

%Open the GAMBIT journal file to write
fid = fopen(strcat('C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder
    \', num2str(sample_nمبر), 'gambit.jou'), 'w');

%Use the lofs and uofs data to print out the jou file
%Input: k - Scaling of the coordinates
%Input: lofs, uofs
%Output: mesh generating command file that is readable by Gambit

k=1;
lofs_mini=lofs/k;
uofs_mini=uofs/k;
count=0;

%Sample lofs and Create the command
for i=1:(size(lofs_mini,2)/20):size(lofs_mini,2)

```

```

        i=floor(i);
        fprintf(fid, 'vertex create coordinates %6.2f %6.2f 0\n', lofs_mini(:,i));
        count=count+1;
    end
    %Sample the last point on the curve
    fprintf(fid, 'vertex create coordinates %6.2f %6.2f 0\n', lofs_mini(:,size(
        lofs_mini,2)));
    count=count+1;

    %Sample uofs and Create the command
    for i=1:(size(uofs_mini,2)/20):size(uofs_mini,2)
        i=floor(i);
        fprintf(fid, 'vertex create coordinates %6.2f %6.2f 0\n', uofs_mini(:,i));
    end
    %Sample the last point on the curve
    fprintf(fid, 'vertex create coordinates %6.2f %6.2f 0\n', uofs_mini(:,size(
        uofs_mini,2)));

    fprintf(fid,'edge create "lofs" nurbs ');
    for i=1:count
        fprintf(fid,'"vertex.%d" ',i);
    end
    fprintf(fid,'interpolate\n');

    fprintf(fid,'edge create "uofs" nurbs ');
    for i=(count+1):(count)*2
        fprintf(fid,'"vertex.%d" ',i);
    end
    fprintf(fid,'interpolate\n');

    fprintf(fid,'edge create "inlet" straight "vertex.1" "vertex.22"\n');
    fprintf(fid,'edge create "outlet" straight "vertex.21" "vertex.42"\n');
    fprintf(fid,'face create "Face1" wireframe "lofs" "uofs" "inlet" "outlet" real\n
        ');
    fprintf(fid,'blayer create first 1.2 growth 1.2 rows 1 transition 1 trows 0
        uniform\n');
    fprintf(fid,'blayer attach "b_layer.1" face "Face1" "Face1" "Face1" "Face1" edge
        "uofs" "lofs" "inlet" "outlet" add\n');
    %    fprintf(fid,'blayer create first 1 growth 1.2 total 9.92992 rows 6
        transition 1 trows 0 uniform\n');
    %    fprintf(fid,'blayer attach "b_layer.1" face "Face1" "Face1" "Face1" "Face1"
        edge "uofs" "lofs" add\n');

    %    fprintf(fid,'face mesh "Face1" map size 2.35\n'); %Specify grid size
    fprintf(fid,'face mesh "Face1" triangle size 3\n'); %use triangle mesh

    fprintf(fid,'physics create btype "WALL" edge "lofs" "uofs"\n');
    fprintf(fid,'physics create btype "VELOCITY_INLET" edge "inlet"\n');
    fprintf(fid,'physics create btype "OUTFLOW" edge "outlet"\n');
    fprintf(fid,'physics create ctype "FLUID" face "Face1"\n');
    fprintf(fid,'export fluent5 ');
    fwrite(fid,'\n');
    fprintf(fid,'\n ');
    fwrite(fid,'"C:\\Documents and Settings\\dl9v07\\Desktop\\CFD-working-folder\\');
    fprintf(fid, strcat(num2str(sample_nmbr),'.msh' nozval'));
    fclose(fid);

```

```

str=['C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\' num2str(
    sample_nmbr) '.msh'];
if exist(str,'file')
    delete (str);
end
str=['C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\' num2str(
    sample_nmbr) '.trn'];
if exist(str,'file')
    delete (str);
end
str=['C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\' num2str(
    sample_nmbr) '.dbs'];
if exist(str,'file')
    delete (str);
end
str=['C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\' num2str(
    sample_nmbr) '.jou'];
if exist(str,'file')
    delete (str);
end
str=['C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\' num2str(
    sample_nmbr) '.lok'];
if exist(str,'file')
    delete (str);
end
str=['C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\' num2str(
    sample_nmbr)];
if exist(str,'file')
    delete (str);
end

%prepare the GAMBIT batch file
fid = fopen('C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\GAMBIT.
    bat','w');
fwrite(fid,['C:\Fluent.Inc\ntbin\ntx86\gambit.exe -r2.4.6 ' num2str(sample_nmbr)
    ' -inputfile C:\Docume~1\dl9v07\Desktop\CFD-W0~1\' num2str(sample_nmbr) '
    gambit.jou']));
fclose(fid);
%Run GAMBIT
winopen('C:\Docume~1\dl9v07\Desktop\CFD-W0~1\GAMBIT.bat')

%Try to find the mesh file, if the geometry fails, there will not be a mesh
str=['C:\Documents and Settings\dl9v07\Desktop\CFD-working-folder\' num2str(
    sample_nmbr) '.msh'];
timecount=0.5;
while ~exist(str,'file')&&timecount<40
    pause(0.5)
    timecount=timecount+0.5;
end
pause(1)
mesh_flag=exist(str,'file');

if mesh_flag==2
    %Prepare the fluent journal file
    fid = fopen(strcat('C:\Documents and Settings\dl9v07\Desktop\CFD-working-
        folder\' ,num2str(sample_nmbr),'.fluent.jou'), 'wt');
    fwrite(fid, ['file/read-case "C:\Documents and Settings\dl9v07\Desktop\CFD-
        working-folder\' num2str(sample_nmbr) '.msh"']);

```

```

fprintf(fid, '\n');
fwrite(fid, 'grid/scale');
fprintf(fid, '\n0.001 \n0.001 \n');
fwrite(fid, 'define/models/viscous');
fprintf(fid, '\nke-standard \nyes \nq\n');
fwrite(fid, 'solve/set/discretization-scheme/pressure');
fprintf(fid, '\n12\n');
fwrite(fid, 'solve/set/discretization-scheme/mom');
fprintf(fid, '\n1\n');
fwrite(fid, 'solve/set/discretization-scheme/epsilon');
fprintf(fid, '\n1\n');
fwrite(fid, 'solve/set/discretization-scheme/k');
fprintf(fid, '\n1\n');
fwrite(fid, 'solve/monitors/residual/convergence-criteria');
fprintf(fid, '\n1e-6\n0.001\n0.001\n0.001\n0.001\n\n');
fwrite(fid, 'define/materials/change-create');
fprintf(fid, '\nair\nair_incompressible\ny\nconstant\n0.38\nno\nno\nny\n\n\n\n\nno\nno\nno\nno\nno\nnyes\nconstant\n295.4\nyes\n\n\n');
fwrite(fid, 'define/operating-conditions');
fprintf(fid, '\noperating-pressure\n23800\nq\n');
fwrite(fid, 'define/boundary-conditions/');
fprintf(fid, '\nvelocity-inlet\n\n\n\n\n\n200\n\n288.15\nnn\n\n\nny\n5\n0\n\n\n\n\n');
fwrite(fid, 'solve/initialize/set-defaults');
fprintf(fid, '\nx-velocity\n200\ntemperature\n288.15\nq\n');
fwrite(fid, 'solve/monitors/residual');
fprintf(fid, '\nplot\ny\nq\n');
fwrite(fid, 'solve/iterate');
fprintf(fid, '\n10\n\nadapt\nadapt-to-y+\n30\n100\n\n\nny\nq\n');
fwrite(fid, 'solve/iterate');
fprintf(fid, '\n10\nplot\nplot\nyes\n'); %number of iterations
fprintf(fid, num2str(sample_nمبر));
fprintf(fid, '\nno\nny\n0\n1\nno\nno\ntotal-pressure\n3\n\n'); %Write to a x-y
data file
%      fprintf(fid, 'q\nexit\nyes\n'); %This line controls whether Fluent
automatically quit after the computation.
fclose(fid);

%Prepare a Fluent batch file
fid = fopen('C:\Documents and Settings\dl9v07\desktop\CFD-working-folder\
FLUENT.bat', 'w');
fwrite(fid, ['C:\Fluent.Inc\ntbin\ntx86\fluent.exe -r6.3.26 2d -i C:\Docume~1\
dl9v07\Desktop\CFD-working-folder\' num2str(sample_nمبر) 'fluent.jou']);
fclose(fid);

%Run Fluent by the batch file
winopen('C:\Documents and Settings\dl9v07\desktop\CFD-working-folder\FLUENT.
bat')

%Read xy data file, find the std of the pressure distribution
str=['C:\Documents and Settings\dl9v07\desktop\CFD-working-folder\' num2str(
sample_nمبر)];
while ~exist(str, 'file')
    pause(0.5)
end
pause(1)
pressure_profile_data=importxy(str);
std_data(sample_nمبر,1)=std(pressure_profile_data(:,1));
else

```

---

```

std_data(sample_nmbr,1)=NaN;
fprintf('No mesh file found for Geometry %d\n',sample_nmbr)
end
close

```

---

LISTING A.12: MATLAB code as described in Figure 5.1

## A.6 Sample GAMBIT journal file

---

```

vertex create coordinates -99.83 283.50 0
vertex create coordinates -66.93 288.67 0
vertex create coordinates -44.23 296.46 0
vertex create coordinates -31.15 300.47 0
vertex create coordinates -29.36 300.89 0
vertex create coordinates -28.79 300.24 0
vertex create coordinates -16.31 284.47 0
vertex create coordinates 1.67 253.89 0
vertex create coordinates 21.49 216.78 0
vertex create coordinates 42.96 179.21 0
vertex create coordinates 68.63 145.21 0
vertex create coordinates 102.68 121.16 0
vertex create coordinates 138.28 113.28 0
vertex create coordinates 166.23 115.40 0
vertex create coordinates 184.58 118.68 0
vertex create coordinates 191.97 119.38 0
vertex create coordinates 197.40 117.86 0
vertex create coordinates 211.47 111.42 0
vertex create coordinates 233.38 101.28 0
vertex create coordinates 263.07 91.65 0
vertex create coordinates 299.30 87.50 0
vertex create coordinates -100.10 371.00 0
vertex create coordinates -91.62 372.61 0
vertex create coordinates -73.80 378.81 0
vertex create coordinates -46.99 386.52 0
vertex create coordinates -9.26 386.06 0
vertex create coordinates 29.62 365.40 0
vertex create coordinates 56.73 332.65 0
vertex create coordinates 78.51 295.75 0
vertex create coordinates 98.56 258.22 0
vertex create coordinates 117.04 225.78 0
vertex create coordinates 131.37 206.19 0
vertex create coordinates 137.26 201.54 0
vertex create coordinates 141.18 200.74 0
vertex create coordinates 151.97 201.73 0
vertex create coordinates 171.16 205.15 0
vertex create coordinates 199.70 206.53 0
vertex create coordinates 228.74 199.55 0
vertex create coordinates 249.93 190.02 0
vertex create coordinates 266.89 182.11 0
vertex create coordinates 282.47 176.97 0
vertex create coordinates 299.50 175.00 0
edge create "lofs" nurbs "vertex.1" "vertex.2" "vertex.3" "vertex.4" "vertex.5" "
vertex.6" "vertex.7" "vertex.8" "vertex.9" "vertex.10" "vertex.11" "vertex
.12" "vertex.13" "vertex.14" "vertex.15" "vertex.16" "vertex.17" "vertex.18"
"vertex.19" "vertex.20" "vertex.21" interpolate

```

---

```

edge create "uofs" nurbs "vertex.22" "vertex.23" "vertex.24" "vertex.25" "vertex
.26" "vertex.27" "vertex.28" "vertex.29" "vertex.30" "vertex.31" "vertex.32"
"vertex.33" "vertex.34" "vertex.35" "vertex.36" "vertex.37" "vertex.38" "
vertex.39" "vertex.40" "vertex.41" "vertex.42" interpolate
edge create "inlet" straight "vertex.1" "vertex.22"
edge create "outlet" straight "vertex.21" "vertex.42"
face create "Face1" wireframe "lofs" "uofs" "inlet" "outlet" real
blayer create first 1.2 growth 1.2 rows 1 transition 1 trows 0 uniform
blayer attach "b_layer.1" face "Face1" "Face1" "Face1" "Face1" edge "uofs" "lofs"
"inlet" "outlet" add
face mesh "Face1" triangle size 3
physics create btype "WALL" edge "lofs" "uofs"
physics create btype "VELOCITY_INLET" edge "inlet"
physics create btype "OUTFLOW" edge "outlet"
physics create ctype "FLUID" face "Face1"
export fluent5 \
"C:\\Documents and Settings\\dl9v07\\Desktop\\CFD-working-folder\\196.msh"
nozval

```

---

LISTING A.13: Sample GAMBIT journal file

## A.7 Sample FLUENT journal file

---

```

file/read-case "C:\\Documents and Settings\\dl9v07\\Desktop\\CFD-working-folder\\196.msh"
grid/scale
0.001
0.001
define/models/viscous
ke-standard
yes
q
solve/set/discretization-scheme/pressure
12
solve/set/discretization-scheme/mom
1
solve/set/discretization-scheme/epsilon
1
solve/set/discretization-scheme/k
1
solve/monitors/residual/convergence-criteria
1e-6
0.001
0.001
0.001
0.001

define/materials/change-create
air
air_incompressible
y
constant
0.38
no
no
y
sutherland

```

```
no
no
no
no
no
yes
constant
295.4
yes

define/operating-conditions
operating-pressure
23800
q
define/boundary-conditions /
velocity-inlet


200

288.15
n


y
5
0.0875
q
solve/initialize/set-defaults
x-velocity
200
temperature
288.15
q
solve/monitors/residual
plot
y
q
solve/iterate
1000

adapt
adapt-to-y+
30
100


y
q
solve/iterate
1000
```

---

```

plot
plot
yes
196
no
y
0
1
no
no
total-pressure
3

q
exit
yes

```

---

LISTING A.14: Sample FLUENT journal file

## A.8 Turbine blade model repair automation

### A.8.1 Sample blade design variable file

---

```

TURBINE BLADE DESIGN VARIABLES
DELTA X:
FRONT_CORE -0.99
DELTA Y:
FRONT_CORE --0.2
DELTA X:
MIDDLE_CORE --0.99
DELTA Y:
MIDDLE_CORE -0.4
DELTA X:
TRAILING_CORE -0.3
DELTA Y:
TRAILING_CORE -0.6
First Extrusion Directions:
X_DIR -0.6
Y_DIR -0.6
Z_DIR --0.6
Second Extrusion Directions:
X_DIR -0.6
Y_DIR -0.6
Z_DIR --0.6
Trailing Hole:
X_up -6.5
Y_up -3
X_down -8
Y_down -0.8

```

---

LISTING A.15: Sample blade design variable file

### A.8.2 NX Open C file that reads blade design variables



---

```

#include <stdio.h>
#include <uf.h>
#include <uf_ui.h>

/*      Function to read in the design variables to construct the engine */
int read_design_vars(double *VARS)
//double *VARS;
{
    int i=0, SUCCESS=1;
    char REPORT[300],s[300];
    char *MAIN_DIR;
    /*char MAIN_DIR []="C:\\Documents and Settings\\dl9v07\\My Documents\\
Visual Studio 2008\\Projects\\C_exe_Project";*/
    char VAR_FILE[300];
    FILE *file;
    double VAR;

    UF_UI_write_listing_window("\n Reading Turbine Blade Design Parameters\n
");

    /* Extract the main directory from the environment variables */
    MAIN_DIR=getenv("TURBINE_BLADE");

    /* Define the full path to the folder containing the design variables*/
    sprintf(VAR_FILE,"%s\\Blade_Variables.exp",MAIN_DIR);

    /* open the .exp file containing the variables */
    file=fopen(VAR_FILE,"r");
    /* terminate the script if the variables file has not been found */
    if (file==NULL){
        UF_UI_write_listing_window("\n Parameters File Not Found\n");
        return(SUCCESS=0);
    }

    /* Import Fan Casing Design Variables */
    fgets(s,300,file); fgets(s,300,file); /* discard header */

    UF_UI_write_listing_window("\t\tDelta X (mm):\t\tDelta Y (mm):\n");

    if (fgets(s,300,file)!=NULL){
        sscanf(s,"FRONT_CORE-%lf",&VAR);
        sprintf(REPORT,"Front Core:\t\t %8.6f",VAR);
        UF_UI_write_listing_window(REPORT);
        *(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }

    fgets(s,300,file); /* discard header */

    if (fgets(s,300,file)!=NULL){
        sscanf(s,"FRONT_CORE-%lf",&VAR);
        sprintf(REPORT,"\t\t%8.6f\n",VAR); UF_UI_write_listing_window(
REPORT);
        *(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }

    UF_UI_write_listing_window("\n");

```

```

/*****
fgets(s,300,file); /* discard header */

if (fgets(s,300,file)!=NULL){
    sscanf(s,"MIDDLE_CORE-%lf",&VAR);
    sprintf(REPORT,"Middle Core:\t %8.6f",VAR);
UF_UI_write_listing_window(REPORT);*(VARS+i++)=VAR;}
else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }

fgets(s,300,file); /* discard header */

if (fgets(s,300,file)!=NULL){
    sscanf(s,"MIDDLE_CORE-%lf",&VAR);
    sprintf(REPORT,"\t\t%8.6f\n",VAR); UF_UI_write_listing_window(
REPORT);*(VARS+i++)=VAR;}
else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }

UF_UI_write_listing_window("\n");

/*****
fgets(s,300,file); /* discard header */

if (fgets(s,300,file)!=NULL){
    sscanf(s,"TRAILING_CORE-%lf",&VAR);
    sprintf(REPORT,"Trailing Core:\t %8.6f",VAR);
UF_UI_write_listing_window(REPORT);*(VARS+i++)=VAR;}
else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }

fgets(s,300,file); /* discard header */

if (fgets(s,300,file)!=NULL){
    sscanf(s,"TRAILING_CORE-%lf",&VAR);
    sprintf(REPORT,"\t\t%8.6f\n",VAR); UF_UI_write_listing_window(
REPORT);*(VARS+i++)=VAR;}
else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }

UF_UI_write_listing_window("\n");

/*****
fgets(s,300,file); /* discard header */

UF_UI_write_listing_window("First Extrusion Direction:\n");
if (fgets(s,300,file)!=NULL){
    sscanf(s,"X_DIR-%lf",&VAR);
    sprintf(REPORT,"\tX: %8.6f",VAR); UF_UI_write_listing_window(
REPORT);*(VARS+i++)=VAR;}
else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
if (fgets(s,300,file)!=NULL){
    sscanf(s,"Y_DIR-%lf",&VAR);
    sprintf(REPORT,"\t\tY: %8.6f",VAR); UF_UI_write_listing_window(
REPORT);*(VARS+i++)=VAR;}
else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }

```

```

    if (fgets(s,300,file)!=NULL){
        sscanf(s,"Z_DIR-%lf",&VAR);
        sprintf(REPORT,"\t\tZ:  %8.6f\n",VAR); UF_UI_write_listing_window
(REPORT);*(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
    UF_UI_write_listing_window("\n");
    /*****

fgets(s,300,file); /* discard header */

UF_UI_write_listing_window("Second Extrusion Direction:\n");
if (fgets(s,300,file)!=NULL){
    sscanf(s,"X_DIR-%lf",&VAR);
    sprintf(REPORT,"\tX:  %8.6f",VAR); UF_UI_write_listing_window(
REPORT);*(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
    if (fgets(s,300,file)!=NULL){
        sscanf(s,"Y_DIR-%lf",&VAR);
        sprintf(REPORT,"\t\tY:  %8.6f",VAR); UF_UI_write_listing_window(
REPORT);*(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
    if (fgets(s,300,file)!=NULL){
        sscanf(s,"Z_DIR-%lf",&VAR);
        sprintf(REPORT,"\t\tZ:  %8.6f\n",VAR); UF_UI_write_listing_window
(REPORT);*(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
    UF_UI_write_listing_window("\n");
    /*****

fgets(s,300,file); /* discard header */

UF_UI_write_listing_window("Two pairs of pole coordinates to determine
the shape of the trailing hole:\n");
if (fgets(s,300,file)!=NULL){
    sscanf(s,"X_up-%lf",&VAR);
    sprintf(REPORT,"\tX_up:\t\t%8.6f",VAR);
UF_UI_write_listing_window(REPORT);*(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
    if (fgets(s,300,file)!=NULL){
        sscanf(s,"Y_up-%lf",&VAR);
        sprintf(REPORT,"\tY_up:\t\t%8.6f\n",VAR);
UF_UI_write_listing_window(REPORT);*(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
    if (fgets(s,300,file)!=NULL){
        sscanf(s,"X_down-%lf",&VAR);
        sprintf(REPORT,"\tX_down:\t\t%8.6f",VAR);
UF_UI_write_listing_window(REPORT);*(VARS+i++)=VAR;}
    else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
    if (fgets(s,300,file)!=NULL){
        sscanf(s,"Y_down-%lf",&VAR);
        sprintf(REPORT,"\tY_down:\t\t%8.6f\n",VAR);
UF_UI_write_listing_window(REPORT);*(VARS+i++)=VAR;}

```

---

```

        else{UF_UI_write_listing_window("\tVariables Mismatch\n"); return(SUCCESS
=0); }
        UF_UI_write_listing_window("\n");

        /* close the file */
        fclose(file);

        return(SUCCESS);
}

```

---

LISTING A.16: NX Open C file that reads blade design variables

### A.8.3 NX Open C file that creates a turbine blade 3D section

---

```

/* Function which creates a 3D section of the turbine blade */

#include <uf_modl.h>
#include <uf_ui.h>
#include <math.h>
#include <uf_obj.h>
#include <uf_curve.h>
#include <uf.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int Blade_Gen(double *VARS)
{
    int SUCCESS=1;
    /* define the NX line structure and tags */
    tag_t spline1_ID, spline2_ID, spline3_ID, spline4_ID, spline5_ID,
    spline6_ID, spline7_ID, spline8_ID; //typedef unsigned int tag_t;
    tag_t spline9_ID, spline10_ID, spline11_ID;
    /* define the spline data structures */
    int n_states, i=0;
    UF_CURVE_state_t *spline_states;
    double knots1[8], poles1[4][4], knots2[13], poles2[9][4], knots3[8],
    poles3[4][4], knots4[15], poles4[11][4];
    double knots5[10], poles5[6][4], knots6[11], poles6[7][4], knots7[12],
    poles7[8][4], knots8[12], poles8[8][4];
    double knots9[12], poles9[8][4], knots10[11], poles10[7][4], knots11[11],
    poles11[7][4]; //trailing edge cooling hole splines
    UF_CURVE_spline_t spline_data1, spline_data2, spline_data3, spline_data4,
    spline_data5, spline_data6;
    UF_CURVE_spline_t spline_data7, spline_data8, spline_data9, spline_data10,
    spline_data11;

    /*define distance variables*/
    double min_dist_4_6;
    double min_dist_2_5;
    double min_dist_4_8;
    double min_dist_2_7;
    double min_dist_4_10;
    double min_dist_2_9;
    double min_dist_6_7;
    double min_dist_8_9;

```

```

double guess[3]={0,0,0};
double pt_on_obj1[3];
double pt_on_obj2[3];

FILE *stream;
int numclosed;
errno_t err;

char REPORT[MAX_TEXT_LENGTH];
/* define the variables */
double FRONT_X=*(VARS+i++);
double FRONT_Y=*(VARS+i++);
double MIDDLE_X=*(VARS+i++);
double MIDDLE_Y=*(VARS+i++);
double TRAILING_X=*(VARS+i++);
double TRAILING_Y=*(VARS+i++);
double Direction_X=*(VARS+i++);
double Direction_Y=*(VARS+i++);
double Direction_Z=*(VARS+i++);
double Direction_X2=*(VARS+i++);
double Direction_Y2=*(VARS+i++);
double Direction_Z2=*(VARS+i++);
double X_up=*(VARS+i++);
double Y_up=*(VARS+i++);
double X_down=*(VARS+i++);
double Y_down=*(VARS+i++);

/* define extrusion data structures */
int obj_cnt;
char *limits[2]={ "0.0", "11" };/*limit[2] is a pointer array. limit[0]
is the address of the first character of the string "0.0". limit[1] is the
address of the string "1"
char *limits_middle[2]={ "0.0", "21" };/*limit[2] is a pointer array.
limit[0] is the address of the first character of the string "0.0". limit[1]
is the address of the string "1"
char *offsets[2]={ "0.0", "0.0" };//ditto
double region_point[3]={0.0,0.0,0.0};
double direction[3]={Direction_X,Direction_Y,Direction_Z};
UF_FEATURE_SIGN mode_sign = UF_NULLSIGN; //typedef enum UF_FEATURE_SIGNS
UF_FEATURE_SIGN; UF_NULLSIGN = 0, /* create new target solid */ this line
declares and assigns the variable mode_sign at the same time
tag_t *Blade_Feat, Blade_Body, extrude_objs[9]; //define pointers of type
tag_t (unsigned interger)
//The term "tag" is used instead of the old term "EID" (Entity ID), so
basically tag is an entity ID.

int obj_cnt2;
double direction2[3]={Direction_X2,Direction_Y2,Direction_Z2};
UF_FEATURE_SIGN mode_sign2 = UF_NEGATIVE; //typedef enum UF_FEATURE_SIGNS
UF_FEATURE_SIGN; UF_NULLSIGN = 0, /* create new target solid */ this
line declares and assigns the variable mode_sign at the same time
tag_t *Blade_Feat2, Blade_Body2, extrude_objs2[2]; //define pointers of
type tag_t (unsigned interger)
//The term "tag" is used instead of the old term "EID" (Entity ID), so
basically tag is an entity ID.

/* define structures to search for and apply tags */
uf_list_p_t face_list, edge_list;

```

```

    int face_count, facetype, edge_count, edge_type, j=0, vertex_count,
    tag_count;
    tag_t faceID, edgeID;
    double edge_point1[3], edge_point2[3];

    UF_UI_write_listing_window("Generating Modified Turbine Blade:\n");

    /* create spline 1 (leading edge) */
    spline_data1.num_poles = 4;
    spline_data1.order = 4;
    spline_data1.is_rational = false;
    spline_data1.start_param = 0.0;
    spline_data1.end_param = 1.0;
    spline_data1.knots=knots1;
    spline_data1.poles=poles1;
    /* define the knots */
    i=0;
    spline_data1.knots[i++]=0.0;
    spline_data1.knots[i++]=0.0;
    spline_data1.knots[i++]=0.0;
    spline_data1.knots[i++]=0.0;
    spline_data1.knots[i++]=1.0;
    spline_data1.knots[i++]=1.0;
    spline_data1.knots[i++]=1.0;
    spline_data1.knots[i++]=1.0;
    /* define the poles */
    i=0;
    spline_data1.poles[i][0]=-10.95; spline_data1.poles[i][1]=-2.70;
    spline_data1.poles[i][2]=0.0; spline_data1.poles[i++][3]=1.0;
    spline_data1.poles[i][0]=-10.9; spline_data1.poles[i][1]=-2.6;
    spline_data1.poles[i][2]=0.0; spline_data1.poles[i++][3]=1.0;
    spline_data1.poles[i][0]=-10.8; spline_data1.poles[i][1]=-2.35;
    spline_data1.poles[i][2]=0.0; spline_data1.poles[i++][3]=1.0;
    // spline_data1.poles[i][0]=-10.7; spline_data1.poles[i][1]=-2.15;
    spline_data1.poles[i][2]=0.0; spline_data1.poles[i++][3]=1.0;
    spline_data1.poles[i][0]=-10.5; spline_data1.poles[i][1]=-2.0;
    spline_data1.poles[i][2]=0.0; spline_data1.poles[i++][3]=1.0;
    /* create the spline */
    UF_CURVE_create_spline(&spline_data1,&spline1_ID,&n_states,&spline_states
);//UF_CURVE_spline_t spline_data1;
    UF_free(spline_states);

    UF_UI_write_listing_window("\t\tLeading edge constructed\n");

    /* create spline 2 (lower surface) */
    spline_data2.num_poles = 9;
    spline_data2.order = 4;
    spline_data2.is_rational = false;
    spline_data2.start_param = 0.0;
    spline_data2.end_param = 1.0;
    spline_data2.knots=knots2;
    spline_data2.poles=poles2;
    /* define the knots */
    i=0;
    spline_data2.knots[i++]=0.0000000000000000;
    spline_data2.knots[i++]=0.0000000000000000;
    spline_data2.knots[i++]=0.0000000000000000;

```

```

    spline_data2.knots[i++]=0.0000000000000000;
    spline_data2.knots[i++]=0.1667;
    spline_data2.knots[i++]=0.3333;
    spline_data2.knots[i++]=0.5;
    spline_data2.knots[i++]=0.6667;
    spline_data2.knots[i++]=0.8333;
    spline_data2.knots[i++]=1.0000000000000000;
    spline_data2.knots[i++]=1.0000000000000000;
    spline_data2.knots[i++]=1.0000000000000000;
    spline_data2.knots[i++]=1.0000000000000000;
    /* define the poles */
    i=0;
    spline_data2.poles[i][0]=-10.5; spline_data2.poles[i][1]=-2.0;
    spline_data2.poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    spline_data2.poles[i][0]=-9.8; spline_data2.poles[i][1]=-1.6;
    spline_data2.poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    spline_data2.poles[i][0]=-7.9; spline_data2.poles[i][1]=-1.6;
    spline_data2.poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    spline_data2.poles[i][0]=-3.5; spline_data2.poles[i][1]=-0.9;
    spline_data2.poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    spline_data2.poles[i][0]=5.5; spline_data2.poles[i][1]=2.9; spline_data2.
    poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    spline_data2.poles[i][0]=9.75; spline_data2.poles[i][1]=6.8; spline_data2
    .poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    spline_data2.poles[i][0]=13.5; spline_data2.poles[i][1]=12.9;
    spline_data2.poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    spline_data2.poles[i][0]=14.0; spline_data2.poles[i][1]=13.8;
    spline_data2.poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    spline_data2.poles[i][0]=14.5; spline_data2.poles[i][1]=14.0;
    spline_data2.poles[i][2]=0.0; spline_data2.poles[i+1][3]=1.0;
    /* create the spline */
    UF_CURVE_create_spline(&spline_data2,&spline2_ID,&n_states,&spline_states
);
    UF_free(spline_states);
    UF_UI_write_listing_window("\t\tLower surface constructed\n");

    /* create spline 3 (trailing edge) */
    spline_data3.num_poles = 4;
    spline_data3.order = 4;
    spline_data3.is_rational = false;
    spline_data3.start_param = 0.0;
    spline_data3.end_param = 1.0;
    spline_data3.knots=knots3;
    spline_data3.poles=poles3;
    /* define the knots */
    i=0;
    spline_data3.knots[i++]=0.0; spline_data3.knots[i+1]=0.0;
    spline_data3.knots[i+2]=0; spline_data3.knots[i+3]=0;
    spline_data3.knots[i+4]=1.0; spline_data3.knots[i+5]=1.0; spline_data3
    .knots[i+6]=1.0; spline_data3.knots[i+7]=1.0;
    /* define the poles */
    i=0;
    spline_data3.poles[i][0]=14.5; spline_data3.poles[i][1]=14.0;
    spline_data3.poles[i][2]=0.0; spline_data3.poles[i+1][3]=1.0;
    spline_data3.poles[i][0]=14.65; spline_data3.poles[i][1]=14.15;
    spline_data3.poles[i][2]=0.0; spline_data3.poles[i+1][3]=1.0;
    spline_data3.poles[i][0]=14.85; spline_data3.poles[i][1]=14.0;
    spline_data3.poles[i][2]=0.0; spline_data3.poles[i+1][3]=1.0;

```

```

        spline_data3.poles[i][0]=14.9; spline_data3.poles[i][1]=13.8;
spline_data3.poles[i][2]=0.0; spline_data3.poles[i++][3]=1.0;
/* create the spline */
UF_CURVE_create_spline(&spline_data3,&spline3_ID,&n_states,&spline_states
);
UF_free(spline_states);
UF_UI_write_listing_window("\t\tTrailing edge constructed\n");

/* create spline 4 (upper surface) */
spline_data4.num_poles = 11;
spline_data4.order = 4;
spline_data4.is_rational = false;
spline_data4.start_param = 0.0;
spline_data4.end_param = 1.0;
spline_data4.knots=knots4;
spline_data4.poles=poles4;
/* define the knots */
i=0;
spline_data4.knots[i++]=0.0;
spline_data4.knots[i++]=0.0;
spline_data4.knots[i++]=0.0;
spline_data4.knots[i++]=0.0;
spline_data4.knots[i++]=0.1250;
spline_data4.knots[i++]=0.2500;
spline_data4.knots[i++]=0.3750;
spline_data4.knots[i++]=0.5000;
spline_data4.knots[i++]=0.6250;
spline_data4.knots[i++]=0.7500;
spline_data4.knots[i++]=0.8750;
spline_data4.knots[i++]=1.0;
spline_data4.knots[i++]=1.0;
spline_data4.knots[i++]=1.0;
spline_data4.knots[i++]=1.0;
/* define the poles */
i=0;
spline_data4.poles[i][0]=14.9; spline_data4.poles[i][1]=13.8;
spline_data4.poles[i][2]=0.0; spline_data4.poles[i++][3]=1.0;
spline_data4.poles[i][0]=14.6; spline_data4.poles[i
][1]=12.9154132218400000; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
++][3]=1.0;
spline_data4.poles[i][0]=11.2; spline_data4.poles[i
][1]=5.0596844425069900; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
++][3]=1.0;
spline_data4.poles[i][0]=4.7; spline_data4.poles[i
][1]=-6.9894388807489900; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
++][3]=1.0;
spline_data4.poles[i][0]=1.3; spline_data4.poles[i
][1]=-10.2025330099099000; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
++][3]=1.0;
spline_data4.poles[i][0]=-2.0; spline_data4.poles[i
][1]=-11.3931509861200000; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
++][3]=1.0;
spline_data4.poles[i][0]=-5.0; spline_data4.poles[i
][1]=-11.1303171135899000; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
++][3]=1.0;
spline_data4.poles[i][0]=-8.5; spline_data4.poles[i
][1]=-8.8109391377529900; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
++][3]=1.0;

```



```

        spline_data4.poles[i][0]=-10.7; spline_data4.poles[i]
    ][1]=-5.0758833937009900; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
    ++][3]=1.0;
        spline_data4.poles[i][0]=-11.0; spline_data4.poles[i]
    ][1]=-3.3139874899890000; spline_data4.poles[i][2]=0.0; spline_data4.poles[i
    ++][3]=1.0;
        spline_data4.poles[i][0]=-10.95; spline_data4.poles[i][1]=-2.70;
    spline_data4.poles[i][2]=0.0; spline_data4.poles[i++][3]=1.0;
        /* create the spline */
        UF_CURVE_create_spline(&spline_data4,&spline4_ID,&n_states,&spline_states
    );
        UF_free(spline_states);
        UF_UI_write_listing_window("\t\tUpper surface constructed\n");

        /* create spline 5 (front cooling core section 1) */
        spline_data5.num_poles = 6;
        spline_data5.order = 4;
        spline_data5.is_rational = false;
        spline_data5.start_param = 0.0;
        spline_data5.end_param = 1.0;
        spline_data5.knots=knots5;
        spline_data5.poles=poles5;
        /* define the knots */
        i=0;
        spline_data5.knots[i++]=0.0;
        spline_data5.knots[i++]=0.0;
        spline_data5.knots[i++]=0.0;
        spline_data5.knots[i++]=0.0;
        spline_data5.knots[i++]=0.333;
        spline_data5.knots[i++]=0.666;
        spline_data5.knots[i++]=1.0;
        spline_data5.knots[i++]=1.0;
        spline_data5.knots[i++]=1.0;
        spline_data5.knots[i++]=1.0;
        /* define the poles */

        i=0;
        spline_data5.poles[i][0]=-6.65; spline_data5.poles[i][1]=-3.0;
    spline_data5.poles[i][2]=0.0; spline_data5.poles[i++][3]=1.0;
        spline_data5.poles[i][0]=-8.5; spline_data5.poles[i][1]=-3.1;
    spline_data5.poles[i][2]=0.0; spline_data5.poles[i++][3]=1.0;
        spline_data5.poles[i][0]=-9.1; spline_data5.poles[i][1]=-3.6;
    spline_data5.poles[i][2]=0.0; spline_data5.poles[i++][3]=1.0;
        spline_data5.poles[i][0]=-9.0; spline_data5.poles[i][1]=-5.1;
    spline_data5.poles[i][2]=0.0; spline_data5.poles[i++][3]=1.0;
        spline_data5.poles[i][0]=-7.5; spline_data5.poles[i][1]=-7.7;
    spline_data5.poles[i][2]=0.0; spline_data5.poles[i++][3]=1.0;
        spline_data5.poles[i][0]=-6.45; spline_data5.poles[i][1]=-8.75;
    spline_data5.poles[i][2]=0.0; spline_data5.poles[i++][3]=1.0;

        /* translate spline in X & Y axis */
        for (i=0; i<spline_data5.num_poles; i++)
        {
            /* adjust the X coordinates */
            spline_data5.poles[i][0]=spline_data5.poles[i][0]+FRONT_X;
            /* adjust the Y coordinates */
            spline_data5.poles[i][1]=spline_data5.poles[i][1]+FRONT_Y;
        }
    
```

```

/* create the spline */
UF_CURVE_create_spline(&spline_data5,&spline5_ID,&n_states,&spline_states
);
UF_free(spline_states);
UF_UI_write_listing_window("\t\tFront core section 1 constructed\n");

/* create spline 6 (front cooling core section 2) */
spline_data6.num_poles = 7;
spline_data6.order = 4;
spline_data6.is_rational = false;
spline_data6.start_param = 0.0;
spline_data6.end_param = 1.0;
spline_data6.knots=knots6;
spline_data6.poles=poles6;
/* define the knots */
i=0;
spline_data6.knots[i++]=0.0;
spline_data6.knots[i++]=0.0;
spline_data6.knots[i++]=0.0;
spline_data6.knots[i++]=0.0;
spline_data6.knots[i++]=0.250;
spline_data6.knots[i++]=0.500;
spline_data6.knots[i++]=0.750;
spline_data6.knots[i++]=1.0;
spline_data6.knots[i++]=1.0;
spline_data6.knots[i++]=1.0;
spline_data6.knots[i++]=1.0;
/* define the poles */
i=0;
spline_data6.poles[i][0]=-6.45; spline_data6.poles[i][1]=-8.75;
spline_data6.poles[i][2]=0.0; spline_data6.poles[i++][3]=1.0;
spline_data6.poles[i][0]=-5.1; spline_data6.poles[i][1]=-9.2;
spline_data6.poles[i][2]=0.0; spline_data6.poles[i++][3]=1.0;
spline_data6.poles[i][0]=-4.5; spline_data6.poles[i][1]=-8.6;
spline_data6.poles[i][2]=0.0; spline_data6.poles[i++][3]=1.0;
spline_data6.poles[i][0]=-4.6; spline_data6.poles[i][1]=-7.0;
spline_data6.poles[i][2]=0.0; spline_data6.poles[i++][3]=1.0;
spline_data6.poles[i][0]=-5.4; spline_data6.poles[i][1]=-4.1;
spline_data6.poles[i][2]=0.0; spline_data6.poles[i++][3]=1.0;
spline_data6.poles[i][0]=-6.0; spline_data6.poles[i][1]=-3.1;
spline_data6.poles[i][2]=0.0; spline_data6.poles[i++][3]=1.0;
spline_data6.poles[i][0]=-6.65; spline_data6.poles[i][1]=-3.0;
spline_data6.poles[i][2]=0.0; spline_data6.poles[i++][3]=1.0;
/* translate spline in X & Y axis */
for (i=0; i<spline_data6.num_poles; i++)
{
    /* adjust the X coordinates */
    spline_data6.poles[i][0]=spline_data6.poles[i][0]+FRONT_X;
    /* adjust the Y coordinates */
    spline_data6.poles[i][1]=spline_data6.poles[i][1]+FRONT_Y;
}
/* create the spline */
UF_CURVE_create_spline(&spline_data6,&spline6_ID,&n_states,&spline_states
);
UF_free(spline_states);
UF_UI_write_listing_window("\t\tFront core section 2 constructed\n");

```

```

/*****

    /* create spline 7 (middle cooling core section 1) */
    spline_data7.num_poles = 7;
spline_data7.order = 4;
spline_data7.is_rational = false;
spline_data7.start_param = 0.0;
spline_data7.end_param = 1.0;
    spline_data7.knots=knots7;
    spline_data7.poles=poles7;
    /* define the knots */
    i=0;
    spline_data7.knots[i++]=0.0;
    spline_data7.knots[i++]=0.0;
    spline_data7.knots[i++]=0.0;
    spline_data7.knots[i++]=0.0;
    spline_data7.knots[i++]=0.25;
    spline_data7.knots[i++]=0.5;
    spline_data7.knots[i++]=0.75;
    spline_data7.knots[i++]=1.0;
    spline_data7.knots[i++]=1.0;
    spline_data7.knots[i++]=1.0;
    spline_data7.knots[i++]=1.0;
    /* define the poles */
    i=0;
    spline_data7.poles[i][0]=3.8; spline_data7.poles[i][1]=0.8; spline_data7.
poles[i][2]=0.0; spline_data7.poles[i++][3]=1.0;
    spline_data7.poles[i][0]=-2.5; spline_data7.poles[i][1]=-2.0;
spline_data7.poles[i][2]=0.0; spline_data7.poles[i++][3]=1.0;
    spline_data7.poles[i][0]=-4; spline_data7.poles[i][1]=-3; spline_data7.
poles[i][2]=0.0; spline_data7.poles[i++][3]=1.0;
    spline_data7.poles[i][0]=-3.0; spline_data7.poles[i][1]=-5; spline_data7.
poles[i][2]=0.0; spline_data7.poles[i++][3]=1.0;
    spline_data7.poles[i][0]=-2.5; spline_data7.poles[i][1]=-7.8;
spline_data7.poles[i][2]=0.0; spline_data7.poles[i++][3]=1.0;
    spline_data7.poles[i][0]=-2; spline_data7.poles[i][1]=-8.75; spline_data7
.poles[i][2]=0.0; spline_data7.poles[i++][3]=1.0;
    spline_data7.poles[i][0]=-1.7; spline_data7.poles[i][1]=-8.5;
spline_data7.poles[i][2]=0.0; spline_data7.poles[i++][3]=1.0;
    /* translate spline in X & Y axis */
    for (i=0; i<spline_data7.num_poles; i++)
    {
        /* adjust the X coordinates */
        spline_data7.poles[i][0]=spline_data7.poles[i][0]+MIDDLE_X;
        /* adjust the Y coordinates */
        spline_data7.poles[i][1]=spline_data7.poles[i][1]+MIDDLE_Y;
    }
    /* create the spline */
    UF_CURVE_create_spline(&spline_data7,&spline7_ID,&n_states,&spline_states
);
    UF_free(spline_states);
    UF_UI_write_listing_window("\t\tMiddle core upper and left edge
constructed\n");

*****/

```

```

/*****

/* create spline 8 (middle cooling core section 2) */
spline_data8.num_poles = 8;
spline_data8.order = 4;
spline_data8.is_rational = false;
spline_data8.start_param = 0.0;
spline_data8.end_param = 1.0;
spline_data8.knots=knots8;
spline_data8.poles=poles8;
/* define the knots */
i=0;
spline_data8.knots[i++]=0.0;
spline_data8.knots[i++]=0.0;
spline_data8.knots[i++]=0.0;
spline_data8.knots[i++]=0.0;
spline_data8.knots[i++]=0.2;
spline_data8.knots[i++]=0.4;
spline_data8.knots[i++]=0.6;
spline_data8.knots[i++]=0.8;
spline_data8.knots[i++]=1.0;
spline_data8.knots[i++]=1.0;
spline_data8.knots[i++]=1.0;
spline_data8.knots[i++]=1.0;
/* define the poles */
i=0;

spline_data8.poles[i][0]=-1.7; spline_data8.poles[i][1]=-8.5;
spline_data8.poles[i][2]=0.0; spline_data8.poles[i++][3]=1.0;
spline_data8.poles[i][0]=3; spline_data8.poles[i][1]=-6; spline_data8.
poles[i][2]=0.0; spline_data8.poles[i++][3]=1.0;
spline_data8.poles[i][0]=5.5; spline_data8.poles[i][1]=-2.5; spline_data8.
poles[i][2]=0.0; spline_data8.poles[i++][3]=1.0;
spline_data8.poles[i][0]=6; spline_data8.poles[i][1]=-2.0; spline_data8.
poles[i][2]=0.0; spline_data8.poles[i++][3]=1.0;
spline_data8.poles[i][0]=5.3; spline_data8.poles[i][1]=0; spline_data8.
poles[i][2]=0.0; spline_data8.poles[i++][3]=1.0;
spline_data8.poles[i][0]=4.75; spline_data8.poles[i][1]=0.8; spline_data8.
poles[i][2]=0.0; spline_data8.poles[i++][3]=1.0;
spline_data8.poles[i][0]=4; spline_data8.poles[i][1]=1; spline_data8.
poles[i][2]=0.0; spline_data8.poles[i++][3]=1.0;
spline_data8.poles[i][0]=3.8; spline_data8.poles[i][1]=0.8; spline_data8.
poles[i][2]=0.0; spline_data8.poles[i++][3]=1.0;
/* translate spline in X & Y axis */
for (i=0; i<spline_data8.num_poles; i++)
{
    /* adjust the X coordinates */
    spline_data8.poles[i][0]=spline_data8.poles[i][0]+MIDDLE_X;
    /* adjust the Y coordinates */
    spline_data8.poles[i][1]=spline_data8.poles[i][1]+MIDDLE_Y;
}
/* create the spline */
UF_CURVE_create_spline(&spline_data8,&spline8_ID,&n_states,&spline_states
);
UF_free(spline_states);
UF_UI_write_listing_window("\t\tMiddle core lower and right edge
constructed\n");

```

```

/*****

/*****

    /* create spline 9 (trailing cooling core upper and left section ) */
    spline_data9.num_poles = 8;
spline_data9.order = 4;
spline_data9.is_rational = false;
spline_data9.start_param = 0.0;
spline_data9.end_param = 1.0;
    spline_data9.knots=knots9;
    spline_data9.poles=poles9;
    /* define the knots */
    i=0;
    spline_data9.knots[i++]=0.0;
    spline_data9.knots[i++]=0.0;
    spline_data9.knots[i++]=0.0;
    spline_data9.knots[i++]=0.0;
    spline_data9.knots[i++]=0.2;
    spline_data9.knots[i++]=0.4;
    spline_data9.knots[i++]=0.6;
    spline_data9.knots[i++]=0.8;
    spline_data9.knots[i++]=1.0;
    spline_data9.knots[i++]=1.0;
    spline_data9.knots[i++]=1.0;
    spline_data9.knots[i++]=1.0;
    /* define the poles */
    i=0;

    spline_data9.poles[i][0]=11; spline_data9.poles[i][1]=7.5; spline_data9.
poles[i][2]=0.0; spline_data9.poles[i++][3]=1.0;
    spline_data9.poles[i][0]=8; spline_data9.poles[i][1]=4.8; spline_data9.
poles[i][2]=0.0; spline_data9.poles[i++][3]=1.0;
    spline_data9.poles[i][0]=X_up; spline_data9.poles[i][1]=Y_up;
spline_data9.poles[i][2]=0.0; spline_data9.poles[i++][3]=1.0;
    spline_data9.poles[i][0]=6; spline_data9.poles[i][1]=2; spline_data9.
poles[i][2]=0.0; spline_data9.poles[i++][3]=1.0;
    spline_data9.poles[i][0]=5.8; spline_data9.poles[i][1]=1.8; spline_data9.
poles[i][2]=0.0; spline_data9.poles[i++][3]=1.0;
    spline_data9.poles[i][0]=6; spline_data9.poles[i][1]=1.1; spline_data9.
poles[i][2]=0.0; spline_data9.poles[i++][3]=1.0;
    spline_data9.poles[i][0]=6.1; spline_data9.poles[i][1]=0.6; spline_data9.
poles[i][2]=0.0; spline_data9.poles[i++][3]=1.0;
    spline_data9.poles[i][0]=6.3; spline_data9.poles[i][1]=0.1; spline_data9.
poles[i][2]=0.0; spline_data9.poles[i++][3]=1.0;
    /* translate spline in X & Y axis */
    for (i=0; i<spline_data9.num_poles; i++)
    {
        /* adjust the X coordinates */
        spline_data9.poles[i][0]=spline_data9.poles[i][0]+TRAILING_X;
        /* adjust the Y coordinates */
        spline_data9.poles[i][1]=spline_data9.poles[i][1]+TRAILING_Y;
    }
    /* create the spline */
    UF_CURVE_create_spline(&spline_data9,&spline9_ID,&n_states,&spline_states
);

```

```

        UF_free(spline_states);
        UF_UI_write_listing_window("\t\tTrailing core upper and left edge
constructed\n");

/*****

/*****

        /* create spline 10 (trailing cooling core lower and right section ) */
        spline_data10.num_poles = 7;
spline_data10.order = 4;
spline_data10.is_rational = false;
spline_data10.start_param = 0.0;
spline_data10.end_param = 1.0;
        spline_data10.knots=knots10;
        spline_data10.poles=poles10;
        /* define the knots */
        i=0;
        spline_data10.knots[i++]=0.0;
        spline_data10.knots[i++]=0.0;
        spline_data10.knots[i++]=0.0;
        spline_data10.knots[i++]=0.0;
        spline_data10.knots[i++]=0.25;
        spline_data10.knots[i++]=0.5;
        spline_data10.knots[i++]=0.75;
        spline_data10.knots[i++]=1.0;
        spline_data10.knots[i++]=1.0;
        spline_data10.knots[i++]=1.0;
        spline_data10.knots[i++]=1.0;
        /* define the poles */
        i=0;

        spline_data10.poles[i][0]=6.3; spline_data10.poles[i][1]=0.1;
spline_data10.poles[i][2]=0.0; spline_data10.poles[i++][3]=1.0;
        spline_data10.poles[i][0]=6.9; spline_data10.poles[i][1]=-0.1;
spline_data10.poles[i][2]=0.0; spline_data10.poles[i++][3]=1.0;
        spline_data10.poles[i][0]=7.1; spline_data10.poles[i][1]=0; spline_data10
.poles[i][2]=0.0; spline_data10.poles[i++][3]=1.0;
        spline_data10.poles[i][0]=7.75; spline_data10.poles[i][1]=0.2;
spline_data10.poles[i][2]=0.0; spline_data10.poles[i++][3]=1.0;
        spline_data10.poles[i][0]=X_down; spline_data10.poles[i][1]=Y_down;
spline_data10.poles[i][2]=0.0; spline_data10.poles[i++][3]=1.0;
        spline_data10.poles[i][0]=10; spline_data10.poles[i][1]=4; spline_data10.
poles[i][2]=0.0; spline_data10.poles[i++][3]=1.0;
        spline_data10.poles[i][0]=11.75; spline_data10.poles[i][1]=7.6;
spline_data10.poles[i][2]=0.0; spline_data10.poles[i++][3]=1.0;
        /* translate spline in X & Y axis */
        for (i=0; i<spline_data10.num_poles; i++)
        {
                /* adjust the X coordinates */
                spline_data10.poles[i][0]=spline_data10.poles[i][0]+TRAILING_X;
                /* adjust the Y coordinates */
                spline_data10.poles[i][1]=spline_data10.poles[i][1]+TRAILING_Y;
        }
        /* create the spline */
        UF_CURVE_create_spline(&spline_data10,&spline10_ID,&n_states,&
spline_states);

```

```

    UF_free(spline_states);
    UF_UI_write_listing_window("\t\tTrailing core lower and right edge
constructed\n");

/*****

/*****

    /* create spline 11 (trailing cooling core tip section ) */
    spline_data11.num_poles = 6;
spline_data11.order = 4;
spline_data11.is_rational = false;
spline_data11.start_param = 0.0;
spline_data11.end_param = 1.0;
    spline_data11.knots=knots11;
    spline_data11.poles=poles11;
    /* define the knots */
    i=0;
    spline_data11.knots[i++]=0.0;
    spline_data11.knots[i++]=0.0;
    spline_data11.knots[i++]=0.0;
    spline_data11.knots[i++]=0.0;
    spline_data11.knots[i++]=0.33;
    spline_data11.knots[i++]=0.66;
    spline_data11.knots[i++]=1.0;
    spline_data11.knots[i++]=1.0;
    spline_data11.knots[i++]=1.0;
    spline_data11.knots[i++]=1.0;
    /* define the poles */
    i=0;

    spline_data11.poles[i][0]=11.75; spline_data11.poles[i][1]=7.6;
spline_data11.poles[i][2]=0.0; spline_data11.poles[i++][3]=1.0;
    spline_data11.poles[i][0]=11.88; spline_data11.poles[i][1]=7.8674;
spline_data11.poles[i][2]=0.0; spline_data11.poles[i++][3]=1.0;
    spline_data11.poles[i][0]=12; spline_data11.poles[i][1]=8.1143;
spline_data11.poles[i][2]=0.0; spline_data11.poles[i++][3]=1.0;
    spline_data11.poles[i][0]=11.5; spline_data11.poles[i][1]=7.95;
spline_data11.poles[i][2]=0.0; spline_data11.poles[i++][3]=1.0;
    spline_data11.poles[i][0]=11.25; spline_data11.poles[i][1]=7.725;
spline_data11.poles[i][2]=0.0; spline_data11.poles[i++][3]=1.0;
    spline_data11.poles[i][0]=11; spline_data11.poles[i][1]=7.5;
spline_data11.poles[i][2]=0.0; spline_data11.poles[i++][3]=1.0;//upper edge
connecting
    /* translate spline in X & Y axis */
    for (i=0; i<spline_data11.num_poles; i++)
    {
        /* adjust the X coordinates */
        spline_data11.poles[i][0]=spline_data11.poles[i][0]+TRAILING_X;
        /* adjust the Y coordinates */
        spline_data11.poles[i][1]=spline_data11.poles[i][1]+TRAILING_Y;
    }
    /* create the spline */
    UF_CURVE_create_spline(&spline_data11,&spline11_ID,&n_states,&
spline_states);
    UF_free(spline_states);

```

```

UF_UI_write_listing_window("\t\tTrailing core tip edge constructed\n");

/*****

/* define the splines to extrude */
i=0;
extrude_objs[i++]=spline1_ID;   extrude_objs[i++]=spline2_ID;
extrude_objs[i++]=spline3_ID;   extrude_objs[i++]=spline4_ID;
extrude_objs[i++]=spline5_ID;   extrude_objs[i++]=spline6_ID;
extrude_objs[i++]=spline9_ID;   extrude_objs[i++]=spline10_ID;
extrude_objs[i++]=spline11_ID;

/* extrude the splines to form a pseudo-2D turbine blade section */
if (UF_MODL_create_extrusion(extrude_objs,9,NULL,"0.0",limits,offsets,
region_point,false,true,direction,mode_sign,&Blade_Feat,&obj_cnt)==0)
{
    UF_UI_write_listing_window("\t\tBlade section extruded
successfully\n");
    /* capture the body tag of the blade */
    UF_MODL_ask_feat_body(*Blade_Feat,&Blade_Body);
    SUCCESS=1;
}
else{UF_UI_write_listing_window("\t\tBlade section extrusion failed\n");
SUCCESS=0;}

/*****

/* define the splines to extrude */
i=0;
extrude_objs2[i++]=spline7_ID;   extrude_objs2[i++]=spline8_ID;

/* extrude the splines to form a pseudo-2D turbine blade section */
if (UF_MODL_create_extrusion(extrude_objs2,2,NULL,"0.0",limits_middle,
offsets,region_point,false,true,direction2,mode_sign2,&Blade_Feat2,&obj_cnt2)
==0)
{
    UF_UI_write_listing_window("\t\tMiddle Core extruded successfully
\n");
    /* capture the body tag of the blade */
    UF_MODL_ask_feat_body(*Blade_Feat2,&Blade_Body2);
    SUCCESS=1;
}
else{UF_UI_write_listing_window("\t\tMiddle core section extrusion
failed\n"); SUCCESS=0;}

/*****

/* Tag all construction splines */
UF_OBJ_set_name(spline1_ID,"SPLINE_1");
UF_OBJ_set_name(spline2_ID,"SPLINE_2");
UF_OBJ_set_name(spline3_ID,"SPLINE_3");
UF_OBJ_set_name(spline4_ID,"SPLINE_4");
UF_OBJ_set_name(spline5_ID,"SPLINE_5");
UF_OBJ_set_name(spline6_ID,"SPLINE_6");
UF_OBJ_set_name(spline7_ID,"SPLINE_7");
UF_OBJ_set_name(spline8_ID,"SPLINE_8");
UF_OBJ_set_name(spline9_ID,"SPLINE_9");
UF_OBJ_set_name(spline10_ID,"SPLINE_10");

```



```

UF_OBJ_set_name(spline11_ID,"SPLINE_11");

/* Tag the blade body */
UF_OBJ_set_name(Blade_Body,"BLADE_BODY");
/* extract the faces of the turbine blade */
UF_MODL_ask_body_faces(Blade_Body,&face_list);
/* determine the number of faces */
UF_MODL_ask_list_count(face_list,&face_count);

/* initialise the face tag counter */
tag_count=0;
for (i=0; i<face_count; i++)
{
    /* extract the ith face from the list */
    UF_MODL_ask_list_item(face_list,i,&faceID);
    /* determine the type of face */
    UF_MODL_ask_face_type(faceID,&facettype);
    /* if the edge is not planar then the face is one of the edges
defined by splines */
    if(facettype!=UF_MODL_PLANAR_FACE)
    {
        /* ask the face edges */
        UF_MODL_ask_face_edges(faceID,&edge_list);
        /* define the no. of edges */
        UF_MODL_ask_list_count(edge_list,&edge_count);
        /* extract end points of the first spline in the list*/
        for (j=0; j<edge_count; j++)
        {
            /* extract the jth edge ID */
            UF_MODL_ask_list_item(edge_list,j,&edgeID);
            /* determine the type of edge */
            UF_MODL_ask_edge_type(edgeID,&edge_type);
            /* a linear edge has been located (all are
vertical in this case) */
            if(edge_type==UF_MODL_SPLINE_EDGE)
            {
                /* determine the spline end points */
                UF_MODL_ask_edge_verts(edgeID,edge_point1
,edge_point2,&vertex_count);

                break;
            }
        }

        /* match these points with the splines created previously
*/
        /* Leading edge */
        if(fabs(edge_point1[0]-spline_data1.poles[0][0])<1e-9 ||
fabs(edge_point2[0]-spline_data1.poles[0][0])<1e-9)
        {
            if(fabs(edge_point1[0]-spline_data1.poles[
spline_data1.num_poles-1][0])<1e-9 || fabs(edge_point2[0]-spline_data1.poles[
spline_data1.num_poles-1][0])<1e-9)
            {
                /* Tag the leading edge */
                UF_OBJ_set_name(faceID,"SURFACE_1");
                tag_count++;
            }
        }

        /* Lower Surface */

```

```

        if (fabs(edge_point1[0]-spline_data2.poles[0][0])<1e-9 ||
        fabs(edge_point2[0]-spline_data2.poles[0][0])<1e-9)
        {
            if (fabs(edge_point1[0]-spline_data2.poles[
spline_data2.num_poles-1][0])<1e-9 || fabs(edge_point2[0]-spline_data2.poles[
spline_data2.num_poles-1][0])<1e-9)
            {
                /* Tag the lower surface */
                UF_OBJ_set_name(faceID,"SURFACE_2");
                tag_count++;
            }
        }
        /* Trailing edge */
        if (fabs(edge_point1[0]-spline_data3.poles[0][0])<1e-9 ||
        fabs(edge_point2[0]-spline_data3.poles[0][0])<1e-9)
        {
            if (fabs(edge_point1[0]-spline_data3.poles[
spline_data3.num_poles-1][0])<1e-9 || fabs(edge_point2[0]-spline_data3.poles[
spline_data3.num_poles-1][0])<1e-9)
            {
                /* Tag the trailing edge */
                UF_OBJ_set_name(faceID,"SURFACE_3");
                tag_count++;
            }
        }
        /* Trailing edge */
        if (fabs(edge_point1[0]-spline_data4.poles[0][0])<1e-9 ||
        fabs(edge_point2[0]-spline_data4.poles[0][0])<1e-9)
        {
            if (fabs(edge_point1[0]-spline_data4.poles[
spline_data4.num_poles-1][0])<1e-9 || fabs(edge_point2[0]-spline_data4.poles[
spline_data4.num_poles-1][0])<1e-9)
            {
                /* Tag the trailing edge */
                UF_OBJ_set_name(faceID,"SURFACE_4");
                tag_count++;
            }
        }

        /* The two splines defining each core both have the same
start and end points */
        /* Forward Core */
        if (fabs(edge_point1[0]-spline_data5.poles[0][0])<1e-9 ||
        fabs(edge_point2[0]-spline_data5.poles[0][0])<1e-9)
        {
            if (fabs(edge_point1[0]-spline_data5.poles[
spline_data5.num_poles-1][0])<1e-9 || fabs(edge_point2[0]-spline_data5.poles[
spline_data5.num_poles-1][0])<1e-9)
            {
                /* Tag the trailing edge */
                UF_OBJ_set_name(faceID,"FWD_CORE");
                tag_count++;
            }
        }
    }
    else
    {
        /* determine which of the planar faces is selected */
        /* ask the face edges */

```

```

UF_MODL_ask_face_edges(faceID,&edge_list);
/* extract the 1st edge ID */
UF_MODL_ask_list_item(edge_list,1,&edgeID);
/* extract end points of the first edge in the list*/
UF_MODL_ask_edge_verts(edgeID,edge_point1,edge_point2,&
vertex_count);

if(fabs(edge_point1[2]+0.5)<1e-9)
{
    /* the lower face has been selected */
    UF_OBJ_set_name(faceID,"LOWER_FACE");
    tag_count++;
}
else
{
    /* the upper face has been selected */
    UF_OBJ_set_name(faceID,"UPPER_FACE");
    tag_count++;
}
}

if(tag_count==8)
    {SUCCESS=1; UF_UI_write_listing_window("\t\t All 8 Tags Applied
successfully\n\n");}
else
    {SUCCESS=1; UF_UI_write_listing_window("\t\tSome Tags Applied
successfully \n\n");}

UF_MODL_ask_minimum_dist(spline4_ID,spline6_ID,0,guess,0,guess,&
min_dist_4_6,pt_on_obj1,pt_on_obj2);
UF_MODL_ask_minimum_dist(spline2_ID,spline5_ID,0,guess,0,guess,&
min_dist_2_5,pt_on_obj1,pt_on_obj2);
UF_MODL_ask_minimum_dist(spline4_ID,spline8_ID,0,guess,0,guess,&
min_dist_4_8,pt_on_obj1,pt_on_obj2);
UF_MODL_ask_minimum_dist(spline2_ID,spline7_ID,0,guess,0,guess,&
min_dist_2_7,pt_on_obj1,pt_on_obj2);
UF_MODL_ask_minimum_dist(spline4_ID,spline10_ID,0,guess,0,guess,&
min_dist_4_10,pt_on_obj1,pt_on_obj2);
UF_MODL_ask_minimum_dist(spline2_ID,spline9_ID,0,guess,0,guess,&
min_dist_2_9,pt_on_obj1,pt_on_obj2);
UF_MODL_ask_minimum_dist(spline6_ID,spline7_ID,0,guess,0,guess,&
min_dist_6_7,pt_on_obj1,pt_on_obj2);
UF_MODL_ask_minimum_dist(spline8_ID,spline9_ID,0,guess,0,guess,&
min_dist_8_9,pt_on_obj1,pt_on_obj2);
sprintf(REPORT,"Distance:\t\t %8.6f\n",min_dist_4_6);
UF_UI_write_listing_window(REPORT);

err= fopen_s( &stream, "minimum_distance.txt", "w" );
if(err!=0)
    {sprintf(REPORT,"The file 'minimum_distance.txt' was not opened\n
");
    UF_UI_write_listing_window(REPORT);}
else
    {
        sprintf(REPORT,"The file 'minimum_distance.txt' was
opened\n",min_dist_4_6);
        UF_UI_write_listing_window(REPORT);
        fprintf(stream,"%f\n",min_dist_4_6);
        fprintf(stream,"%f\n",min_dist_2_5);
    }

```

---

```

        fprintf(stream,"%f\n",min_dist_4_8);
        fprintf(stream,"%f\n",min_dist_2_7);
        fprintf(stream,"%f\n",min_dist_4_10);
        fprintf(stream,"%f\n",min_dist_2_9);
        fprintf(stream,"%f\n",min_dist_6_7);
        fprintf(stream,"%f\n",min_dist_8_9);
    }

    if( stream)
    {

        if ( fclose( stream ) )
        {
            sprintf( REPORT, "The file '
minimum_distance.txt' was not closed\n" );
            UF_UI_write_listing_window(REPORT);
        }
    }

    return SUCCESS;
}

```

---

LISTING A.17: NX Open C file that creates a turbine blade 3D section

#### A.8.4 Batch file that automates geometry generation

---

```

title NX Part Generation
if exist "%TURBINE_BLADE%\PART_GEN.log" del "%TURBINE_BLADE%\PART_GEN.log"
if exist "%TURBINE_BLADE%\TURBINE_BLADE.prt" del "%TURBINE_BLADE%\TURBINE_BLADE.
prt"
if exist "%TURBINE_BLADE%\TURBINE_BLADE.igs" del "%TURBINE_BLADE%\TURBINE_BLADE.
igs"
if exist "%TURBINE_BLADE%\TURBINE_BLADE.fbm" del "%TURBINE_BLADE%\TURBINE_BLADE.
fbm"
if exist "%TURBINE_BLADE%\TURBINE_BLADE.pm" del "%TURBINE_BLADE%\TURBINE_BLADE.pm
"
if exist "%TURBINE_BLADE%\TURBINE_BLADE_done.pm" del "%TURBINE_BLADE%\
TURBINE_BLADE_done.pm"
if exist "%TURBINE_BLADE%\TURBINE_BLADE.png" del "%TURBINE_BLADE%\TURBINE_BLADE.
png"
if exist "%TURBINE_BLADE%\sc03.diag" del "%TURBINE_BLADE%\sc03.diag"
if exist "%C:\Docume~1\dl9v07\minimum_distance.txt" del "C:\Docume~1\dl9v07\
minimum_distance.txt"
Set UGII_BASE_DIR="C:\Program Files\UGS\NX 6.0\"
Call %UGII_BASE_DIR%\UGII\ugii.cmd bat %UGII_BASE_DIR% AUTO
"%TURBINE_BLADE%\C_exe_Project.exe" > "%TURBINE_BLADE%\PART_GEN.log"
dir > "%TURBINE_BLADE%\RUN_CHECK.txt"

```

---

LISTING A.18: Batch file that automates geometry generation

#### A.8.5 Batch file that automates IGES format geometry generation

---

```

title IGES Generation
Set UGII_BASE_DIR="C:\Program Files\UGS\NX 6.0\"
Call %UGII_BASE_DIR%\UGII\ugiicmd.bat %UGII_BASE_DIR% AUTO
"%UGII_BASE_DIR%\IGES\iges.cmd" "%TURBINE_BLADE%\TURBINE_BLADE.prt" 0="%TURBINE_BLADE%\TURBINE_BLADE.igs" D="%UGII_BASE_DIR%\IGES\igesexport.def"

```

---

LISTING A.19: Batch file that automates IGES format geometry generation

### A.8.6 Batch file that automates pm format geometry generation

---

```

title pm Generation
cd /d "%TURBINE_BLADE%"
"C:\Program Files\CADfix 7.1\bin\runcadconsole.exe" -wait -BATCH -config SC03.cwc TURBINE_BLADE.igs -exportfile TURBINE_BLADE.pm
dir > "%TURBINE_BLADE%\RUN_CHECK.txt"

```

---

LISTING A.20: Batch file that automates pm format geometry generation

### A.8.7 Batch file that automatically starts SC03 and loads SC03 executable file

---

```

title Dong Li's SC03 analysis
cd /d "%TURBINE_BLADE%"
C:\DOCUMENT~1\d19v07\Desktop\stress~1\SC03_1~1\bin\Winsc03.exe -b -e "blade.exec"

```

---

LISTING A.21: Batch file that automatically starts SC03 and loads SC03 executable file

### A.8.8 SC03 executable file for setup, meshing, analysis and output

---

```

# Exec file to run SC03 analysis of the 3D turbine blade
# Dong
# 05/10/2010

# Initialization

# no_ct88 stops CT88 material property checks
no_ct88

# Prevent SC03 from stopping scripts or exiting on an error
ignore off

# Read in the required data files
rd PM "C:\Documents and Settings\d19v07\Desktop\stress analysis\Automation\TURBINE_BLADE.pm"

# mat file contains material data
rd MAT "C:\Documents and Settings\d19v07\Desktop\stress analysis\sc03_13b0x32\bin\BLADE_MATERIAL.mat"

```

---

```

rd GBP "C:\Documents and Settings\d19v07\Desktop\stress analysis\sc03_13b0x32\bin
\boundaries.gbp"

# bdd file contains cycle data
rd BDD "C:\Documents and Settings\d19v07\Desktop\stress analysis\sc03_13b0x32\bin
\cycle.bdd"

domxpan 1
domain 1 speed=0
domain 1 th_temp=500;th_units=C;st_temp=20;st_units=C
domain 1 thermal_flag=1
domain 1 elastic=Y
domain 1 int_ht=0
domain 1 therm_output=Y;therm_refinement=Y
domain 1 component=1
domain 1 material=RCY
domain 1 name=D1
global 1 rotation=X
global 1 analtype=TM
global 1 GNL=N
global 1 thermal_strain=Y
global 1 follower_tractions=N
global 1 thermal_bc_press=Y
global 1 GNL_on_1st_iter=Y
global 1 eqn_reform=N
global 1 residual_tol=1.0E-04
global 1 stress_acc=1000000000000
global 1 strain_acc=0.10
global 1 relative_acc=2
global 1 struct_bound_loads=Y
global 1 struct_body_loads=Y
global 1 steadystate=Y
global 1 coupled=Y
global 1 baseparm=N
global 1 matching=N
global 1 viewfactor_calcs=Y
global 1 max_temp=2000
global 1 th_time_acc=10
global 1 th_ref_acc=10

mesh 1
run

index
locate
case time 1000
CTP TK
gsave TURBINE_BLADE.png
QDOM[5]=0
QDOM[1]=1
call scobk
$MIN=QDOM[3]
$MAX=QDOM[5]
$MEAN=QDOM[4]
echo MIN TEMPERATURE $MIN
echo MAX TEMPERATURE $MAX
echo MEAN TEMPERATURE $MEAN

index

```

```

locate
case time 1000
CTP SVM
gsave TURBINE_BLADE.png
QDOM[5]=0
QDOM[1]=1
call scobk
$MIN=QDOM[3]
$MAX=QDOM[5]
$MEAN=QDOM[4]
echo MIN VONMISES $MIN
echo MAX VONMISES $MAX
echo MEAN VONMISES $MEAN

wr PM "C:\Documents and Settings\dl9v07\Desktop\stress analysis\Automation\
TURBINE_BLADE_done.pm"

```

LISTING A.22: SC03 executable file

### A.8.9 SC03 material property file

```

#
# material file produced by SC03 version 10E0 on 20-MAR-07
# from model file C:\abhi1\hiparsysnatfreq\hiparsys1.pm
#
#
# Thermo-elastic properties for RCY
#
RCY
ISO_ORTH
0.872000E-08
64
#
# T (deg C)      E (MPa)      G (Mpa)      v      alpha
#                k (mW/mm/K) Cp (mJ/Mg/K)
#
-100.000      131200.      134050.      0.375500      0.119500E-04
              9.05000      0.360000E+09
-80.0000      130650.      133525.      0.375750      0.119734E-04
              9.27500      0.365000E+09
-60.0000      130100.      133000.      0.376000      0.119975E-04
              9.50000      0.370000E+09
-40.0000      129550.      132475.      0.376250      0.120227E-04
              9.72500      0.375000E+09

...

800.000      98700.0      104000.      0.394000      0.143000E-04
              19.9000      0.560000E+09
820.000      97551.1      103268.      0.395051      0.144200E-04
              20.1400      0.567280E+09
840.000      96372.0      102590.      0.396192      0.145400E-04
              20.3800      0.575840E+09
860.000      95117.3      101918.      0.397408      0.146600E-04
              20.6357      0.583840E+09
880.000      93741.8      101204.      0.398683      0.147800E-04

```

---

	20.9666	0.591280E+09		
900.000	92200.0	100400.	0.400000	0.149000E-04
	21.3000	0.600000E+09		
920.000	90560.0	99310.0	0.401347	0.150104E-04
	21.5598	0.610000E+09		
940.000	88920.0	97878.7	0.402720	0.151272E-04
	21.8235	0.620000E+09		
960.000	87280.0	96272.4	0.404120	0.152800E-04
	22.2000	0.630000E+09		
980.000	85640.0	94657.4	0.405547	0.154400E-04
	22.6000	0.640000E+09		
1000.00	84000.0	93200.0	0.407000	0.156000E-04
	23.0000	0.650000E+09		
1020.00	82360.0	91999.4	0.408520	0.157600E-04
	23.4000	0.661280E+09		
1040.00	80720.0	90848.5	0.410147	0.159200E-04
	23.8000	0.673840E+09		
1060.00	79029.6	89435.2	0.411899	0.160704E-04
	24.2056	0.686000E+09		
1080.00	77051.2	87535.4	0.413864	0.161788E-04
	24.6432	0.698000E+09		
1100.00	74900.0	85300.0	0.416000	0.163000E-04
	25.1000	0.710000E+09		
1120.00	72900.0	83340.0	0.418000	0.164932E-04
	25.5680	0.719920E+09		
1140.00	70900.0	81380.0	0.420000	0.167536E-04
	26.0520	0.726960E+09		
1150.00	69900.0	80400.0	0.421000	0.169000E-04
	26.3000	0.730000E+09		

---

LISTING A.23: SC03 material property file

### A.8.10 3D evolutionary strategy optimisation MATLAB code for proposing turbine blade repair alternative

---

```

% function evop_repair determines a repair alternative for design D0 based on a
% prediction
% function and a penalty threshold.
%
% Input - X0: original design variable set
%         objhandle: name of the prediction function
%         th: the prescribed penalty threshold value
%
% Output - X_best_initial: initial repair
%         - X_best_second: initial circular repair
%         - X_best_final: final suggested repair alternative
%
% SVR_repair calls this function.
% Do not forget to load the surrogate parameters
%
% Created:      10/06/2010
% Last modified: 28/11/2011
% Lastest changes: add initial global search result, X_best_initial, and
%                  X_best_second circular search
% result as outputs.X_best is replaced as X_best_final. Also add two graphs
% to illustrate the generation of X_best_second and X_best_final.

```



```

% Also use i=round(logspace(5,6,100)) to stay within RAM limit
% Terminate when 5 alternatives are found
% Dong Li

function [X_best_initial,X_best_second,X_best_final]=evop_repair_3D(X0,objhandle,
    th)
dimension=size(X0,2);
% Initial global search
for i=round(logspace(3,6,20))%i is a row vector of 20 logarithmically equally
    spaced points between 10-2~10-5
    fprintf('Sample size = %d...\n',i);
    SAMPLE_PLAN=rlh(i,dimension);
    RESULT=zeros(i,2); RESULT(:)=Inf;
    for j=1:i % search x1 in the whole design space
        RESULT(j,1)=norm(X0-SAMPLE_PLAN(j,:)); % Calculate the
        distance
        RESULT(j,2)=feval(objhandle,SAMPLE_PLAN(j,:)); % Calculate the penalty
    end

    [dmin,indexbp,Bplogical]=best_point_3d(RESULT,th);
    if dmin<inf
        X1=SAMPLE_PLAN(indexbp,:);
        X_best_initial=X1;
        X_best_second=X1;
        X_best_final=X1;
        fprintf('A repair alternative has been found by a global search with size
%d, continue with hyper-circle search\n',i)
        X_best_initial
        fprintf('dmin=%d\n',dmin);
        fprintf('Penalty=%d\n',RESULT(indexbp,2));
        figure
        hold on;
        plot(RESULT(:,1),RESULT(:,2),'.');
        plot(RESULT(Bplogical,1), RESULT(Bplogical,2) , 'ro');
        grid on
        xlabel('16-dimensional Euclidean distance from the original design');
        ylabel('Predicted Penalty');
        line(xlim, [th,th]);
        xlimit=xlim;
        text((xlim(1)+xlim(2))/2, th+0.02, strcat('Penalty threshold P_{th
}=',num2str(th,'%1.4f')), 'fontsize',12)
        plot(RESULT(indexbp,1),RESULT(indexbp,2),'x')
        text(RESULT(indexbp,1),RESULT(indexbp,2),strcat(' \leftarrow Selected
point P_{th}=',num2str(RESULT(indexbp,2),'%1.4f'))
        ylimit=ylim;
        ylim([RESULT(indexbp,2)-0.1,ylim(2)])
        drawnow
        hold off
        break
    end
end

if dmin==inf
    fprintf('No repair alternative can be found\n');
    X_best_final=X0;
    return
end

```

```

fprintf('Initial search in the hyper-circle centered at the original design, with
radius==dmin\n')
for i=round(logspace(5,6,100))
    fprintf('Sample size = %d...\n',i);
    SAMPLE_PLAN=rlh(i,dimension);
    for j=1:i
        SAMPLE_PLAN(j,:)=(SAMPLE_PLAN(j,:)-0.5)*dmin/2+X0; %sqrt(dimension)=sqrt
(16)=4;      2/4=1/2
    end
    RESULT=result_3D(SAMPLE_PLAN,X0,dmin,objhandle);
    [d,indexbp,Bplogical]=best_point_3d(RESULT,th);
    Xc=SAMPLE_PLAN(indexbp,:);
    if d<dmin
        X_best_second=Xc;
        X_best_final=Xc;
        dmin=d;
        fprintf('A better repair alternative is found\n');
        X_best_second
        fprintf('dmin=%d\n',dmin);
        fprintf('Penalty=%d\n',RESULT(indexbp,2));
        figure
        hold on;
        plot(RESULT(:,1),RESULT(:,2),'.');
        plot(RESULT(Bplogical,1), RESULT(Bplogical,2) , 'ro');
        grid on
        xlabel('16-dimensional Euclidean distance from the original design');
        ylabel('Predicted Penalty');
        line(xlim, [th,th]);
        xlimit=xlim;
        text((xlimit(1)+xlimit(2))/2, th+0.02, strcat('Penalty threshold P_{th}
')=',num2str(th,'%1.4f')), 'fontsize',12)
        plot(RESULT(indexbp,1),RESULT(indexbp,2),'x')
        text(RESULT(indexbp,1),RESULT(indexbp,2),strcat(' \leftarrow Selected
point P_{th}= ',num2str(RESULT(indexbp,2),'%1.4f'))
        ylimit=ylim;
        ylim([RESULT(indexbp,2)-0.1,ylimit(2)])
        drawnow
        hold off
        break %breaks here so that as soon as a better repair alternative is
found, this loop is terminated
    end
end
if X_best_final==X1
    fprintf('A better repair alternative cannot be found, using X_best_final==X1\n
');
    return
end

fprintf('Continuous search in hypercircle\n');
Xc=zeros(0,dimension);
count=0;
while (count<5)%To find five repair alternatives
    X_best_final=Xc;
    count=count+1;
    for i=round(logspace(5,6,100))
        fprintf('Sample size = %d...\n',i);
        SAMPLE_PLAN=rlh(i,dimension);
        for j=1:i
            SAMPLE_PLAN(j,:)=(SAMPLE_PLAN(j,:)-0.5)*dmin/2+X0;

```

---

```

end
RESULT=result_3D(SAMPLE_PLAN,X0,dmin,objhandle);
[d,indexbp,Bplogical]=best_point_3d(RESULT,th);
Xc=SAMPLE_PLAN(indexbp,:);
if d<dmin
    X_best_final=Xc;
    dmin=d;
    fprintf('dmin=%d\n',dmin);
    fprintf('Penalty=%d\n',RESULT(indexbp,2));
    fprintf('A better repair alternative is found\n');
    X_best_final
    fprintf('dmin=%d\n',dmin);
    fprintf('Penalty=%d\n',RESULT(indexbp,2));
    figure
    hold on;
    plot(RESULT(:,1),RESULT(:,2),'.');
    plot(RESULT(Bplogical,1), RESULT(Bplogical,2) , 'ro');
    grid on
    xlabel('16-dimensional Euclidean distance from the original design');
    ylabel('Predicted Penalty');
    line(xlim, [th,th]);
    xlimit=xlim;
    text((xlim(1)+xlim(2))/2, th+0.02, strcat('Penalty threshold P_{
th}=',num2str(th,'%1.4f')), 'fontsize',12)
    plot(RESULT(indexbp,1),RESULT(indexbp,2),'x')
    text(RESULT(indexbp,1),RESULT(indexbp,2),strcat(' \leftarrow Selected
point P_{th}=',num2str(RESULT(indexbp,2),'%1.4f')))
    ylim=ylim;
    ylim([RESULT(indexbp,2)-0.1,ylim(2)])
    drawnow
    hold off
    break
end
end
end

```

---

LISTING A.24: 3D evolutionary strategy optimisation MATLAB code for proposing turbine blade repair alternative

### A.8.11 MATLAB code for stress analysis

---

```

function [error_flag, max_temp, max_stress, max_distortion] =
    stress_analysis_assembly(front_core_x,front_core_y,middle_core_x,
        middle_core_y,trailing_core_x,trailing_core_y,x_dir,y_dir,z_dir,X_DIR,Y_DIR,
        Z_DIR,X_up,Y_up,X_down,Y_down)
% This function assembles all analysis components.
% Input: All blade design variables
% Output: One of the failure mode flags, or the resulting maximum temperature and
% maximum von mises stress
% error_flag=0; Successful
% error_flag=1; NX Part generation failure
% error_flag=2; IGES generation failure
% error_flag=3; pm generation failure
% error_flag=4; result generation failure
% Dong Li 04/11/2010

%Initialization

```

```

error_flag=0;

%Write a Blade_Variable.exp file using all the input variables
fid = fopen('C:\Documents and Settings\d19v07\Desktop\stress analysis\Automation\
    Blade_Variables.exp', 'wt');
fprintf(fid,'TURBINE BLADE DESIGN VARIABLES\n');
fprintf(fid,'DELTA X:\n');
fprintf(fid, strcat('FRONT_CORE-',num2str(front_core_x)));
fprintf(fid,'\nDELTA Y:\n');
fprintf(fid, strcat('FRONT_CORE-',num2str(front_core_y)));
fprintf(fid,'\nDELTA X:\n');
fprintf(fid, strcat('MIDDLE_CORE-',num2str(middle_core_x)));
fprintf(fid,'\nDELTA Y:\n');
fprintf(fid, strcat('MIDDLE_CORE-',num2str(middle_core_y)));
fprintf(fid,'\nDELTA X:\n');
fprintf(fid, strcat('TRAILING_CORE-',num2str(trailing_core_x)));
fprintf(fid,'\nDELTA Y:\n');
fprintf(fid, strcat('TRAILING_CORE-',num2str(trailing_core_y)));
fprintf(fid,'\nFirst Extrusion Directions:\n');
fprintf(fid, strcat('X_DIR-',num2str(x_dir)));
fprintf(fid,'\n');
fprintf(fid, strcat('Y_DIR-',num2str(y_dir)));
fprintf(fid,'\n');
fprintf(fid, strcat('Z_DIR-',num2str(z_dir)));
fprintf(fid,'\nSecond Extrusion Directions:\n');
fprintf(fid, strcat('X_DIR-',num2str(X_DIR)));
fprintf(fid,'\n');
fprintf(fid, strcat('Y_DIR-',num2str(Y_DIR)));
fprintf(fid,'\n');
fprintf(fid, strcat('Z_DIR-',num2str(Z_DIR)));
fprintf(fid,'\nTrailing Hole:\n');
fprintf(fid, strcat('X_up-',num2str(X_up)));
fprintf(fid,'\n');
fprintf(fid, strcat('Y_up-',num2str(Y_up)));
fprintf(fid,'\n');
fprintf(fid, strcat('X_down-',num2str(X_down)));
fprintf(fid,'\n');
fprintf(fid, strcat('Y_down-',num2str(Y_down)));
fclose(fid);

%Generate the turbine model by calling NX6 in batch mode.
system('C:\Documents and Settings\d19v07\Desktop\stress analysis\Automation\1
    _TURBINE_GEN.bat');
%Try to find the UG part file, if the geometry generation fails, there will not
    be a part file
str='C:\Documents and Settings\d19v07\Desktop\stress analysis\Automation\
    TURBINE_BLADE.prt';

%Leave 20 seconds for the UG part to be generated.
count=0;
while ((~exist(str,'file'))&&(count<200))
    pause(0.1)
    count=count+1;
end
part_flag=exist(str,'file');
if part_flag~=2
    %Part file does not exist
    error_flag=1; %Part generation failure
    max_temp=NaN;

```

```

        max_stress=NaN;
        max_distortion=NaN;
        return
    end

%Generate the iges format of the blade model using NX6 in batch mode
system('C:\Documents and Settings\dl9v07\Desktop\stress analysis\Automation\2
    _IGES_GEN.bat');
str='C:\Documents and Settings\dl9v07\Desktop\stress analysis\Automation\
    TURBINE_BLADE.igs';
count=0;
while ((~exist(str,'file'))&&(count<200))
    pause(0.1)
    count=count+1;
end
iges_flag=exist(str,'file');
if iges_flag~=2
    %IGES file does not exist
    error_flag=2; %IGES generation failure
    max_temp=NaN;
    max_stress=NaN;
    max_distortion=NaN;
    return
end

%Generate the pm file, which can be read by Rolls-Royce SC03, using CADFix
%in batch mode
system('C:\Documents and Settings\dl9v07\Desktop\stress analysis\Automation\3
    _PM_GEN.bat');
str='C:\Documents and Settings\dl9v07\Desktop\stress analysis\Automation\
    TURBINE_BLADE.pm';
count=0;
while ((~exist(str,'file'))&&(count<60))
    pause(1)
    count=count+1;
end
pm_flag=exist(str,'file');
if pm_flag~=2
    %pm file does not exist
    error_flag=3; %pm generation failure
    max_temp=NaN;
    max_stress=NaN;
    max_distortion=NaN;
    return
end

%Run SC03. Generate the sc03 result files
system('C:\Documents and Settings\dl9v07\Desktop\stress analysis\Automation\4
    _SC03_RUN.bat');
str='C:\Documents and Settings\dl9v07\Desktop\stress analysis\Automation\
    TURBINE_BLADE_done.pm';
count=0;
%Leave 40 seconds for the pm result file (TURBINE_BLADE_done.pm) to be generated.
while ((~exist(str,'file'))&&(count<400))
    pause(0.1)
    count=count+1;
end
result_flag=exist(str,'file');
```

```
if result_flag~=2
    %result file does not exist
    error_flag=4; %result generation failure
    max_temp=NaN;
    max_stress=NaN;
    max_distortion=NaN;
    return
end

%Read maximum temperature and maximum von mises stress from sc03.diag
Diag_ID=fopen('C:\Docume~1\d19v07\Desktop\stress~1\Automa~1\sc03.diag');
A=fread(Diag_ID,'uint8=>char');
A=A';
temp_index=strfind(A,'MAX TEMPERATURE');
temp=A(temp_index+16:temp_index+21);
max_temp=str2double(temp);

stress_index=strfind(A,'MAX VONMISES');
von_mises=A(stress_index+13:stress_index+18);
max_stress=str2double(von_mises);

distortion_index=strfind(A,'Largest distortion measure');
distortion=A(distortion_index+27:distortion_index+34);
max_distortion=str2double(distortion);

fclose(Diag_ID);
```

LISTING A.25: MATLAB code for stress analysis



# References

- A. Agarwal and B. Triggs. Recovering 3d human pose from monocular images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):44 – 58, 2006. ISSN 01628828.
- A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.
- T. Bäck, U. Hammel, and H.P. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1): 3–17, 1997.
- A.D. Belegundu and S.D. Rajan. Shape optimization approach based on natural design variables and shape functions. *Computer Methods in Applied Mechanics and Engineering*, 66(1):87 – 106, 1988. ISSN 00457825.
- H. Beyer and H. Schwefel. Evolution strategies a comprehensive introduction. *Natural Computing*, 1:3–52, 2002. ISSN 1567-7818. 10.1023/A:1015059928466.
- P.E. Bézier. How renault uses numerical control for car body design and tooling. Technical report, Society of Automotive Engineers, 400 Commonwealth Dr, Warrendale, PA, 15096, USA, 1968.
- R.E. Bixby. The simplex method: it keeps getting better. In *Lecture at the 14th International Symposium on Mathematical Programming*, Amsterdam, The Netherlands, 1991.
- G. Bloch, F. Lauer, G. Colin, and Y. Chamaillard. Support vector regression from simulation data and few experimental samples. *Information Sciences*, 178(20):3813 – 3827, 2008. ISSN 00200255.
- M.I.G. Bloor and M.J. Wilson. Generating blend surfaces using partial differential equations. *Computer-Aided Design*, 21:165–171, April 1989.
- P.T. Boggs and J.W. Tolle. Sequential quadratic programming. *Acta numerica*, 4:1–51, 2008.



- A.P. Boresi and R. J. Schmidt. *Advanced mechanics of materials*. John Wiley & Sons, 6 edition, 2003.
- B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.
- G.E.P. Box and K.B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(1):1 – 45, 1951.
- C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121– 67, 1998. ISSN 1384-5810.
- G. Butlin and C. Stops. Cad data repair. In *Proceedings of the 5th International Meshing Roundtable*, pages 7–12, 1996.
- C.S. Chong, A. Senthil Kumar, and H.P. Lee. Automatic mesh-healing technique for model repair and finite element model generation. *Finite Elements in Analysis and Design*, 43(15):1109 – 1119, 2007. ISSN 0168-874X.
- S.M. Clarke, J.H. Griebisch, and T.W. Simpson. Analysis of support vector regression for approximation of complex engineering analyses. *Journal of Mechanical Design*, 127(6):1077–87, Nov. 2005. ISSN 0738-0666.
- A.R. Conn, N.I.M. Gould, P.L. Toint, and P.L. Toint. *Trust-region methods*. Society for Industrial Mathematics, 2000.
- C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- N. Cressie. The origins of kriging. *Mathematical Geology*, 22(3):239 – 252, 1990. ISSN 08828121.
- N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines (and other kernel-based learning methods)*. Cambridge University Press, 2000. ISBN 0 521 78019 5.
- G.B. Dantzig and Wm. Orchard-Hays. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, 8(46):64–67, April 1954. ISSN 08916837.
- M.A. Day. The no-slip condition of fluid dynamics. *Erkenntnis*, 33:285–296, 1990. ISSN 0165-0106. 10.1007/BF00717588.
- C. de Boor. Cadre: An algorithm for numerical quadrature. *Mathematical software*, pages 417–449, 1971.
- J. Dongarra and F. Sullivan. Guest editors’ introduction: The top 10 algorithms. *Computing in Science and Engineering*, 2:22–23, 2000. ISSN 1521-9615.

- H.B. Drucker, C.J.C. Kaufman, L. Smola, and V.A. Vapnik. Support vector regression machines. *Advances in Neural Information Processing Systems*, (9):155–161, 1997.
- F. Duchaine, T. Morel, and L.Y.M. Gicquel. Computational-fluid-dynamics-based kriging optimization tool for aeronautical combustion chambers. *AIAA Journal*, 47(3):631 – 645, 2009. ISSN 00011452.
- S.B. Eom. Expert system applications in production and operations management: A selected bibliography (1975-1989). *Expert Systems with Applications*, 5(12):167–183, 1992.
- R. Fletcher and M.J.D Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163, 1963.
- FLUENT. *FLUENT 6.3 User's Guide*, chapter 12.11.1 Near-Wall Mesh Guidelines. Fluent Inc., Lebanon, USA, 2006a.
- FLUENT. *FLUENT 6.3 User's Guide*, chapter 12 Modeling Turbulence. Fluent Inc., Lebanon, USA, 2006b.
- A.I.J. Forrester, A. Sóbester, and A. J. Keane. *Engineering Design via Surrogate Modeling: A Practical Guide*. Wiley, Chichester, 2008.
- K. Fukuda and T. Terlaky. Criss-cross methods: A fresh view on pivot algorithms. *Mathematical Programming, Series B*, 79(1-3):369 – 395, 1997. ISSN 00255610.
- T.S. Furey, N. Cristianini, N. Duffy, D.W. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.
- D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675.
- S.H. Gopalakrishnan and K. Suresh. A formal theory for estimating defeaturing-induced engineering analysis errors. *Computer Aided Design*, 39(1):60–68, 1 2007. ISSN 0010-4485.
- S.R. Gunn. Support vector machines for classification and regression. Technical report, University of Southampton, 1998.
- C. He, Y. Li, Y. Huang, C. Liu, and S. Fei. Relevance vector machine based gear fault detection. pages 731 – 735, Nanjing, China, 2009. Empirical results; Fault diagnosis; Gear fault detection; Generalization performance; Kernel function; Machine availability; Maintenance cost; On-line fault detection; Relevance Vector Machine; Testing time; Training time;.

- M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.
- J.H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.
- N. Hoyle, N.W. Bressloff, and A.J. Keane. Design optimization of a two-dimensional subsonic engine air intake. *AIAA Journal*, 44(11):2672 – 2681, 2006. ISSN 00011452.
- C.W Hsu, C.C Chang, and C.J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2003.
- J.T. Hu, Y.K. Lee, T. Blacker, and J. Zhu. Overlay grid based geometry cleanup. In *11th International Meshing Roundtable*, Ithaca, New York, USA., 9 2002.
- D. L. Hughes, L. P. Myers, and K.G.Mackall. Effects of inlet distortion on a static pressure probe mounted on the engine hub in an f-15 airplane. Technical report, NASA Ames Research Center, 1985.
- E. N. Jacobs, K. E. Ward, and R. M. Pinkerton. The characteristics of 78 related airfoil sections from tests in the variable-density wind tunnel. Report 460, NACA, 1933.
- J. Jian. A new feasible descent algorithm combining sqp with generalized projection for optimization problems without strict complementarity. *Applied Mathematics and Computation (New York)*, 162(3):1065 – 1081, 2005. ISSN 00963003.
- T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Machine Learning: ECML-98*, volume 1398 of *Lecture Notes in Computer Science*, pages 137–142. Springer Berlin / Heidelberg, 1998. ISBN 978-3-540-64417-0.
- M.E. Johnson, L.M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, 26(2):131–148, 1990.
- D.R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345 – 383, 2001. ISSN 0925-5001.
- I. C. Karpolis, E. I. Karangelos, and K. C. Giannakoglou. Gradient-assisted radial basis function networks: Theory and applications. *Applied Mathematical Modelling*, 28(2): 197 – 209, 2004. ISSN 0307-904X. surrogate modelling.
- N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.
- W. Karush. Minima of functions of several variables with inequalities as side constraints. Master’s thesis, Dept. of mathematics, University of Chicago, 1939.

- A. J. Keane. Statistical improvement criteria for use in multiobjective design optimization. *AIAA Journal*, 44(4):879 – 91, 04 2006. ISSN 0001-1452.
- A. J. Keane and P. B. Nair. *Computational Approaches for Aerospace Design*, chapter 2, page 56. Wiley, 2005a. ISBN 9780470855409.
- A. J. Keane and P. B. Nair. *Computational Approaches for Aerospace Design*. Wiley, 2005b. ISBN 9780470855409.
- A. J. Keane and P. B. Nair. *Computational Approaches for Aerospace Design*, chapter 3, page 147. Wiley, 2005c. ISBN 9780470855409.
- A.J. Keane and J.P. Scanlan. Design search and optimization in aerospace engineering. *Philosophical Transactions of the Royal Society London, Series A (Mathematical, Physical and Engineering Sciences)*, 365(1859):2501 – 29, 10 2007. ISSN 1364-503X.
- L.G. Khachian. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20:53–72, 1980.
- S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5):975–986, 1984.
- A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering and System Safety*, 91(9):992 – 1007, 2006. ISSN 09518320.
- J.C. Korngold and G.A. Gabriele. Multidisciplinary analysis and optimization of discrete problems using response surface methods. *Journal of Mechanical Design, Transactions of the ASME*, 119(4):427 – 433, 1997. ISSN 1050-0472.
- G. La Rocca and M.J.L. van Tooren. Knowledge-based engineering to support aircraft multidisciplinary design and optimization. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 224(9):1041–1055, 2010.
- F. Lauer and G. Bloch. Incorporating prior knowledge in support vector regression. *Machine Learning*, 70(1):89–118, January 2008.
- B.E. Launder and D. B. Spalding. *Lectures in Mathematical Models of Turbulence*. Academic Press, London, England, 1972.
- C.T. Lawrence. *A computationally efficient feasible sequential quadratic programming algorithm*. PhD thesis, University of Maryland, 1998.
- S.J. Leary, A. Bhaskar, and A.J. Keane. A knowledge-based approach to response surface modelling in multifidelity optimization. *Journal of Global Optimization*, 26(3):297 – 319, 2003. ISSN 0925-5001.

- K.Y. Lee, M. A. Price, C. G. Armstrong, M.G. Larson, and K. Samuelsson. Cad-to-cae integration through automated model simplification and adaptive modelling. In *Proceedings of the International Conference on Adaptive Modeling and Simulation*, 2003.
- J.P. Leiva and B.C. Watson. Shape optimization in the genesis program. *Optimization in Industry*, 2:219 – 225, 1999. GENESIS program;Shape optimization;.
- K. Lewis and F. Mistree. Collaborative, sequential, and isolated decisions in design. *Journal of Mechanical Design, Transactions of the ASME*, 120(4):643 – 652, 1998. ISSN 1050-0472.
- M.L. Maher, D. Sriram, and S.J. Fenves. Tools and techniques for knowledge based expert systems for engineering design. *Advances in Engineering Software*, 6(4):178 – 188, 1984. ISSN 0141-1195.
- G. Matheron. Principles of geostatistics. *Economic Geology*, 58:1246–1266, 1963.
- M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.
- M. Meckesheimer, R.R. Barton, T. Simpson, F. Limayen, and B. Yannou. Metamodeling of combined discrete/continuous responses. *AIAA Journal*, 39(10):1950 – 1959, 2001. ISSN 0001-1452.
- M.D. Morris and T.J. Mitchell. Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, 43(3):381–402, 1995.
- NASA. Examining spatial (grid) convergence. Retrived 02/04/2011, from <http://www.grc.nasa.gov/WWW/wind/valid/tutorial/spatconv.html>, 07 2008a.
- NASA. Uncertainty and error in cfd simulations. Retrived 12/04/2011, from <http://www.grc.nasa.gov/WWW/wind/valid/tutorial/errors.html>, 07 2008b.
- NASA. Inlet performance. Retrived 02/04/2011, from <http://www.grc.nasa.gov/WWW/K-12/airplane/inleth.html>, 08 2009.
- NASA. Earth atmosphere model. Retrived 12/05/2011, from <http://www.grc.nasa.gov/WWW/K-12/airplane/atmos.html>, 07 2010.
- A. Ostermeier, A. Gawelczyk, and N. Hansen. Step-size adaptation based on non-local use of selection information. In *Parallel Problem Solving from Nature PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 189–198. Springer Berlin / Heidelberg, 1994. ISBN 978-3-540-58484-1.
- R.M. Oxman and R.E. Oxman. Formal knowledge in knowledge-based cad. *Building and Environment*, 26(1):35 – 40, 1991. ISSN 0360-1323.

- K.H. Pan, C.J. Lih, and S.N. Cohen. Analysis of dna microarrays using algorithms that employ rule-based expert knowledge. *Proceedings of the National Academy of Sciences of the United States of America*, 99(4):2118–2123, Feb 19 2002. ISSN 0027-8424.
- R. M. Jr Pickett, M.F. Rubinstein, and R.B. Nelson. Automated structural synthesis using a reduced number of design coordinates. *AIAA Journal*, 11(4):489 – 494, 1973. ISSN 00011452.
- L. Piegl and W. Tiller. *The NURBS book*. Monographs in Visual Communications. Springer-Verlag, New York, NY, second edition, 1997. ISBN 3540615458.
- T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247(4945):978–82, 1990. ISSN 0036-8075. rbf.
- C. Power, J. F. Navarro, A. Jenkins, C. S. Frenk, S. D. M. White, V. Springel, J. Stadel, and T. Quinn. The inner structure of  $\Lambda$ cdm haloes - i. a numerical convergence study. *Monthly Notices of the Royal Astronomical Society*, 338(1):14–34, 2003. ISSN 1365-2966.
- A. Price, A.J. Keane, and C. Holden. On the coordination of multidisciplinary design optimization using expert systems. In *Learning and Intelligent Optimization*, volume 6073 of *Lecture Notes in Computer Science*, pages 212–215. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-13799-0.
- Y. Quan, J. Yang, L. Yao, and C. Ye. An improved way to make large-scale svr learning practical. *EURASIP Journal on Applied Signal Processing*, 8(8):1135–41, July 2004. ISSN 1110-8657.
- I. Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany, 1973.
- Rolls-Royce. *The Jet Engine*. Rolls-Royce, 6 edition, 2005. ISBN 9780902121232.
- R. Roy, S. Hinduja, and R. Teti. Recent advances in engineering design optimisation: Challenges and future trends. *CIRP Annals - Manufacturing Technology*, 57(2):697–715, Apr. 2008. ISSN 00078506.
- G. Rudolph. Global optimization by means of distributed evolution strategies. In *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 209–213. Springer Berlin / Heidelberg, 1991. ISBN 978-3-540-54148-6.
- J. Sacks, W. J. Welch, T. J. Mitchell, and H. P Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.
- J.A. Samareh. Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization. *AIAA Journal*, 39(5):877–884, May 2001.

- J. S. Sánchez, V. García, and R. A. Mollineda. Exploring synergetic effects of dimensionality reduction and resampling tools on hyperspectral imagery data classification. In *Proceedings of the 7th international conference on Machine learning and data mining in pattern recognition*, MLDM'11, pages 511–523, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-23198-8.
- L.A. Schmit Jr. and B. Farshi. Some approximation concepts for structural synthesis. *AIAA Journal*, 12(5):692 – 699, 1974. ISSN 00011452.
- B. Schoelkopf, K. Sung, C.J.C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, 45(11):2758 – 2765, 1997. ISSN 1053587X.
- F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Survey*, 34(1):1–47, March 2002. ISSN 0360-0300.
- J. Seddon and E. L. Goldsmith. *Intake Aerodynamic : an account of the mechanics of flow in and around the air intakes of turbine-engined and ramjet aircraft and missiles*. Professional and Technical Books. Collins,London, 1985. ISBN 0003830489.
- T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. *Computer Graphics (ACM)*, 20(4):151 – 160, 1986. ISSN 00978930.
- J.J. Shah and M. Mantyla. *Parametric and Feature-Based CAD/CAM*. Wiley, New York,, 1995.
- NX Open Programmer's Guide*. Siemens Product Lifecycle Management Software Inc., 2008.
- Timothy W. Simpson and Joaquim R.R.A. Martins. Multidisciplinary design optimization for complex engineered systems: Report from a national science foundation workshop. *Journal of Mechanical Design, Transactions of the ASME*, 133(10), 2011. ISSN 10500472.
- A.J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.
- A. Sobester. Tradeoffs in jet inlet design: a historical perspective. *Journal of Aircraft*, 44(3):705 – 17, 05 2007. ISSN 0021-8669.
- A. Sóbester and A.J. Keane. Supervised learning approach to parametric computer-aided design geometry repair. *AIAA Journal*, 44(2):282–289, Feb. 2006. ISSN 0001-1452.
- J. Sobieszczanski-Sobieski and R.T. Haftka. Multidisciplinary aerospace design optimization: Survey of recent developments. *Structural Optimization*, 14(1):1 – 23, 1997. ISSN 09344373. Multidisciplinary optimization (MDO);.

- W. Song and A.J. Keane. An efficient evolutionary optimisation framework applied to turbine blade firtree root local profiles. *Structural and Multidisciplinary Optimization*, 29(5):382 – 90, 05 2005. ISSN 1615-147X.
- I.E. Sutherland. *Sketchpad: A man-machine graphical communication system*. PhD thesis, Massachusetts Institute of Technology, 1963.
- W. Sutherland. The viscosity of gases and molecular force. *Philosophical Magazine*, 36: 507–531, 1893.
- T. Terlaky. A convergent criss-cross method. *Optimization*, 16(5):683–690, 1985.
- M.E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.
- D.J.J. Toal, N.W. Bressloff, and A.J. Keane. Kriging hyperparameter tuning strategies. *AIAA Journal*, 46(5):1240–1252, 2008.
- R.J. Vanderbei. *Linear programming: foundations and extensions*. Springer Verlag, 2008.
- G.N. Vanderplaats. *Multidiscipline Design Optimization*. Vanderplaats Research & Development, Incorporated, 2007. ISBN 9780944956045.
- V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In *Advances in Neural Information Processing Systems 9*, pages 281–287. MIT Press, 1996.
- V.N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94559-8.
- J.L. Walsh, J.C. Townsend, A.O. Salas, J.A. Samareh, V. Mukhopadhyay, and J.F. Barthelemy. Multidisciplinary high-fidelity analysis and optimization of aerospace vehicles, part 1: Formulation. *AIAA*, 2000.
- F. Wang and Q. Zhang. Knowledge-based neural models for microwave design. *IEEE Transactions on Microwave Theory and Techniques*, 45(12 pt 2):2333 – 2343, 1997. ISSN 00189480. Microwave design;.
- Z.M. Wang. A finite conformal-elimination free algorithm over oriented matroid programming. *Chinese Annals of Mathematics. Series B*, 8(1):120–125, 1987. ISSN 0252-9599. A Chinese summary appears in Chinese Annals of Mathematics. Ser. A 8 (1987), no. 1, 138.
- K. Ye, W. Li, and A. Sudjianto. Algorithmic construction of optimal symmetric latin hypercube designs. *Journal of Statistical Planning and Inference*, 90(1):145–159, 2000.
- S. Yi, J. Shin, and G. Park. Comparison of mdo methods with mathematical examples. *Structural and Multidisciplinary Optimization*, 35:391–402, 2008. ISSN 1615-147X.



- J. Yuan, K. Wang, T. Yu, and X. Liu. Incorporating fuzzy prior knowledge into relevance vector machine regression. In *Proceedings of the International Joint Conference on Neural Networks*, pages 510 – 515, Hong Kong, China, 2008.