June 2017

FITTING A PARAMETRIC MODEL TO A CLOUD OF POINTS VIA OPTIMIZATION METHODS

Pengcheng Jia
Syracuse University

Abstract

Computer Aided Design (CAD) is a powerful tool for designing parametric geometry. However, many CAD models of current configurations are constructed in previous generations of CAD systems, which represent the configuration simply as a collection of surfaces instead of as a parametrized solid model. But since many modern analysis techniques take advantage of a parametrization, one often has to re-engineer the configuration into a parametric model. The objective here is to generate an efficient, robust, and accurate method for fitting parametric models to a cloud of points. The process uses a gradient-based optimization technique, which is applied to the whole cloud, without the need to segment or classify the points in the cloud *a priori*.

First, for the points associated with any component, a variant of the Levenberg-Marquardt gradient-based optimization method (ILM) is used to find the set of model parameters that minimizes the least-square errors between the model and the points. The efficiency of the ILM algorithm is greatly improved through the use of analytic geometric sensitivities and sparse matrix techniques. Second, for cases in which one does not know *a priori* the correspondences between points in the cloud and the geometry model's components, an efficient initialization and classification algorithm is introduced. While this technique works well once the configuration is close enough, it occasionally fails when the initial parametrized configuration is too far from the cloud of points. To circumvent this problem, the objective function is modified, which has yielded good results for all cases tested.

This technique is applied to a series of increasingly complex configurations. The final configuration represents a full transport aircraft configuration, with a wing, fuselage, empennage, and engines. Although only applied to aerospace applications, the technique is general enough to be applicable in any domain for which basic parametrized models are available.

FITTING A PARAMETRIC MODEL TO A CLOUD OF POINTS VIA OPTIMIZATION METHODS

by
Pengcheng Jia

B.S., Tianjin University of Commerce, 2010
M.S., Beijing Jiaotong University, 2012

Dissertation
Submitted in partial fulfillment of the requirement for the degree of
Doctor of Philosophy in Mechanical and Aerospace Engineering

Syracuse University
May 2017

Acknowledgements

I would like to, at this point, acknowledge all of the people who made it possible for me to get to this point. Without the help of these people I would not be who I am today.

First I would like to thank my family, especially my Mother and Father who have supported me unconditionally in everything I have done.

I would also like to thank my advisor, Dr. John Dannenhoffer for working with me for the past three years and always pushing me to do better and be better. He was always available to help, especially early in the morning when no one else was in Link. Thanks to his help and input I have been able to succeed.

I want to thank the members of my committee, Dr. Sinead C. Mac Namara, Dr.Thong Dang, Dr. Utpal Roy, Dr. Melissa Green, and Dr. Benjamin Akih-Kumgeh, for taking time out of their busy schedules to read my thesis, listen to my defense, and provide feedback to improve my research.

I wish to thank Jack Rossetti, Han Lee, Yafei Mei and Yuyan Hao for being my best friends and colleagues in the Aerospace Computational Methods Lab (ACML). They gave me a lot of help and ideas during my research.

I want to thank Ed Alyanak who is the technical monitor of AFRL project. This work was performed as part of the CAPS project, which was funded under AFRL Contract FA8050-14-C-2472: "CAPS: Computational Aircraft Prototype Syntheses".

Pengcheng Jia

Contents

List of Figures

List of Tables

Nomenclature

a, b	super-ellipse radii
c	chord length of NACA airfoil
$cycle$	one (re-)initialization/(re-)classification followed by several iterations
$C1$	inequality constraint
$C2$	equality constraint
C_a	capacity of link a
$\vec{d}$	design parameter vector
d_N	normalized design parameter
d_r	original design parameter
e_i	element in objective function
err	error value of the noisy points in cloud
E	energy state
f_x, f_y	function for generating geometry model
f_p^{rs}	route flow
$\vec{g}$	gradient vector of objective function
H	Hessian matrix
$iter$	each step within the LM optimization algorithm
$IterMax$	the maximum number of iteration
J	Jacobian matrix

K	temperature decrement factor
m	number of points in cloud
m_{NACA}	the maximum camber
n	number of design parameters
p	location of maximum camber
P	Probability function of simulated annealing
q	components of vector to be minimized
q^{rs}	travel demand
r	power in super-ellipse
S	objective function
t	the number of square terms in objective function
t_a	travel time
t_{NACA}	the maximum thickness as a fraction of the chord
T_0	initial temperature
T_f	freezing temperature
T_N	Normalized time
(u_i, v_i)	parametric coordinates
w	weight coefficient of objective function
x_i	position along the chord line
X	design variables in simulated annealing
(x_p, y_p, z_p)	coordinates of points in cloud
(x_t, y_t)	points on super-ellipse
y_t	thickness x_i location
y_c	y coordinate of the camber line
(x_U, y_U)	coordinates of upper airfoil surface
(x_L, y_L)	coordinates of lower airfoil surface

α_a	free-flow travel time
β_L	lower bound of β
β_U	upper bound of β
$\vec{\beta}$	vector of variables to be changed by optimizer $(d_1 \cdots d_n, u_1 \cdots u_m)$
γ	step size in gradient decent method
ν_a	traffic flow
θ	rotation angle along z axis
θ_c	rotated angle of upper/lower airfoil surface's coordinates
η_p^{rs}	route cost function
π^{rs}	minimal route cost
λ	damping factor in LM method
λ_p^{rs}	route-specific cost
$\delta x, \delta y, \delta z$	components of translation
δ	parameter step size of each iteration
Subscript	
i	index of points in cloud
l	index of elements in objective function
j, k	index of parametric model parameters
(s)	index of iterations
ic	index of objective function
jc	index of inequality constraint
kc	index of equality constraint
lc	index of upper and lower bounds of parameter
qc	index of design parameters for example optimization problem
rs	original- destination pair
p	index of route

Chapter 1

Introduction

Computer Aided Design (CAD) tools have evolved over the past several decades from simple 2D drawing tools to parametric design tools in 3D. The use of these models has greatly enhanced the design and analysis process in many fields, such as mechanical manufacturing, CFD, and so on. Unfortunately, many computer-based models of current configurations are constructed in the previous generation of CAD systems, resulting simply in surface definitions. To apply many modern analysis processes to the configuration, it requires that one re-engineer the configuration into a parametric model. For example, for data exchange or manufacturing purposes, the initial CAD parametric data may be unavailable, lost, or no longer corresponds to the original CAD model if the 3D mesh is deformed by another designer or after a numerical simulation process.[3] Moreover, interoperability among heterogeneous CAD systems is also an important issue. The problem stems from the distinct modeling units of each CAD system.[22]

For regenerating a geometric model, freeform parametric surface (NUBRS) fitting to a cloud of point is a fundamental method.[23] However, this method uses a group of discrete surfaces/curves for representing the geometry. It has two disadvantages: because there

are only design parameters for each surface/curve, the effect of design parameters to the features of the whole geometry model (such sensitivity) can not be obtained; and the group of discrete surfaces/curves may not match each other at the connections areas (has gaps between surfaces/curves).

Therefore, for overcoming there disadvantages, in this paper, an efficient, robust and accurate method for generating parametric models from a cloud of point has been developed, using an optimization method applied to the whole data at the same time; there is no need to segment or classify the cloud points.

1.1 Objective

The objective of this work is to find the design parameters associated with a parametric solid model that best-fits a cloud of points, and to do so efficiently and accurately.

Fitting parametric geometry models is a typical problem in many fields, such as: reverse engineering, data exchange, parametric reconstruction, and so on. The fitting of aircraft parametric models requires more efficiency and accuracy due to the large number of points and complicated shapes. These are introduced in Chapter 1.

An optimization technique is used for the fitting process. Gradient-based and heuristics optimization techniques contain many specific algorithms. Based on the different advantages and disadvantages for each method, the new optimization algorithm is developed for fitting the parametric models. It is the combination of gradient-based optimization and heuristics optimization. The main optimization technique and the other application of the new optimization algorithm are explained in Chapter 2.

The new optimization algorithm is applied to fitting a parametric model to a cloud of points. The simplified demonstrations are discussed in Chapter 3. The more general

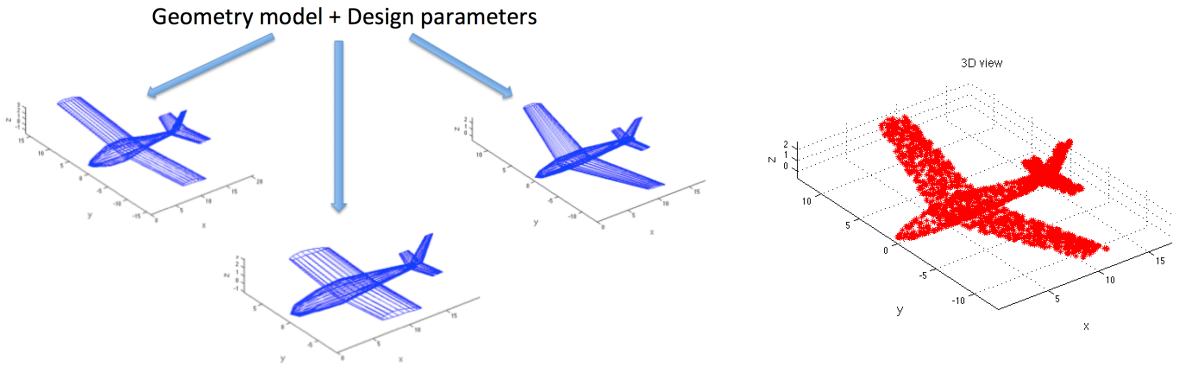
parametric model fitting problems is introduced in Chapter 4. This includes the initialization and classification techniques. For accommodating less accurate initial guesses of the design parameters, the original objective function for optimization is modified. This is also discussed in Chapter 4.

The large number of points (in the cloud) and the many design parameters in aircraft require an efficient way of generating parametric models. A secondary objective is to design a parametric geometry generator that is able to efficiently, robustly, and accurately fit a cloud of points and output geometric parameters. The framework for doing this is discussed in Chapter 5.

Finally Chapter 6 provides a summary, the major conclusions, and suggested areas for future work.

1.2 Parametric Models Generation

As stated, parametric models are often used to analyze aircraft designs. In order to do so, it is important to understand the definition of parametric models, and to survey the different applications in which the generation of parametric models have been successful. Figure 1.1 is the schematic diagram of a parametric aircraft model from a generated from a cloud of points.



(a) Parametric aircraft models

(b) Cloud of Points

Figure 1.1: Example of the parametric models generation

To start, the idea of parametric models must be defined, but a formal definition is difficult to find. In statistics, Bickel and Doksum state [24] that a parametric model or parametric family or finite-dimensional model is a family of distributions that can be described using a finite number of parameters. These parameters are usually collected together to form a single k-dimensional parameter vector $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$. In the geometric field, the parametric model means the features of the geometries. Mantyla and Nau [25] states that features are defined to be parametric shapes associated with attributes such as: intrinsic geometric parameters (length, width, depth etc.), position and orientation, geometric tolerances, material properties, and references to other features. Parametric feature-based modeling is frequently combined with constructive binary solid geometry (CSG) to fully describe systems of complex objects in engineering. In our specific problem, the generation of parametric models is focused on the geometric parameters.

As stated, the generation of parametric geometry models has been applied in many fields, including: reverse engineering [26], data exchange [27], parametric reconstruction [8], geometry fitting [28], and aircraft analysis [29]. The application of parametric models generation is discussed in these fields in the following section.

1.2.1 Reverse Engineering

Generating parametric geometry models is a typical problem in reverse engineering. Before applying the generation of parametric geometry models in reverse engineering, it is important to understand the definition and the contents of reverse engineering. Reverse engineering is accomplished in three steps: part digitizing, features extraction, and CAD modeling. [30]

- Part digitization: acquire point coordinates from real geometry model surfaces
- Features extraction: choose the mathematical functions to model the geometry
- CAD modeling: generate the CAD models based on parametric model

Generally, it starts with measuring an existing object using a laser scanner, and then the measuring data is used to construct a surface or solid model [31]. Although the reverse engineering process may seem to be the opposite of the conventional manufacturing process, in truth the overall concept of the two are quite similar, as shown in Figure 1.2.

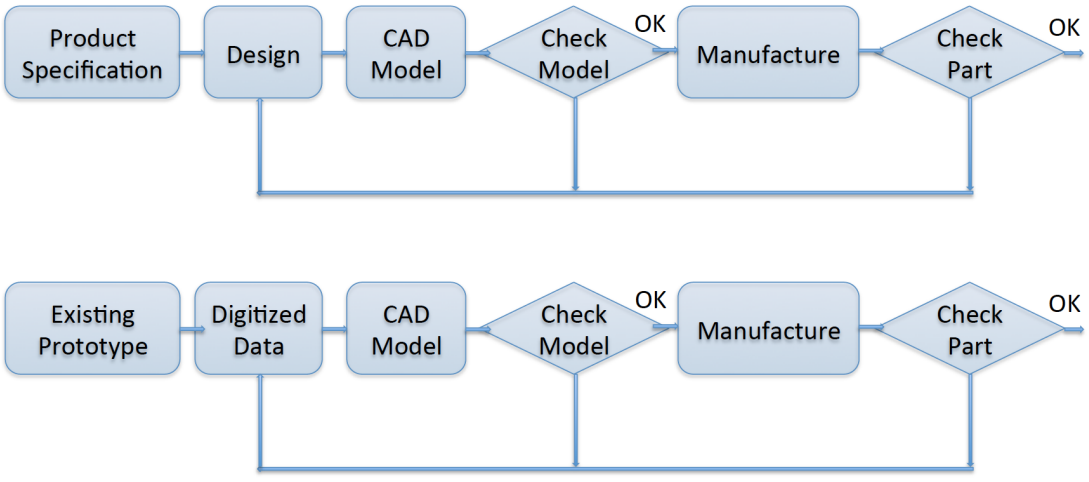


Figure 1.2: Conventional manufacturing and reverse engineering sequences.
(Top) Conventional manufacturing sequence;
(Bottom) reverse engineering manufacturing sequence [1]

Reverse engineering can be applied to complex surface generating and solid modeling. It plays an important role in reconstruction of a surface and significantly reduces the reconstruction time and the costs of the part duplication [2]. There are many cases that one needs reversing engineering to generate parametric models from existing model. Puntambekar [1] summarized five instances for the applications of reverse engineering:

- New Design: The new design starts from a clay model, created either by an artist or iteratively formed through extensive model analysis, e.g. aerodynamic design of a vehicle, gas passages in an engine cylinder block.
- Old Parts Redesign: The drawings for a particular part in a machine may no longer be available, and reverse engineering may be the only way to reproduce such parts, e.g. machines manufactured years ago.
- Worn or Damaged Parts: If a certain machine component is worn out or damaged, and the drawings are not immediately available or are untraceable, the part can be quickly digitized and reverse engineered to restart the production rather than wait for the new component to arrive, e.g. parts on demand.
- Inspection: Samples from a production run can be taken and digitized to construct a CAD model which represents the actual manufactured model, and can be used for inspection purposes. This can be compared against the original CAD model in the database, and the tolerances on the dimensions can also be checked
- Replicating Components: Acquired in this case, the drawings are not being available and such components, if needed, must be reverse engineered.

Zhou [2] presented a new approach to the reconstruction of a surface. The proposed methodology finds the basic parts of the surface and blends surfaces between them. Each basic geometric part is divided into triangular patches that are compared using normal

vectors for face grouping. Each basic geometric surface is then implemented to the infinitive surface. The infinitive surface intersections are trimmed by boundary representation model reconstruction. (such as in Figure 1.3)

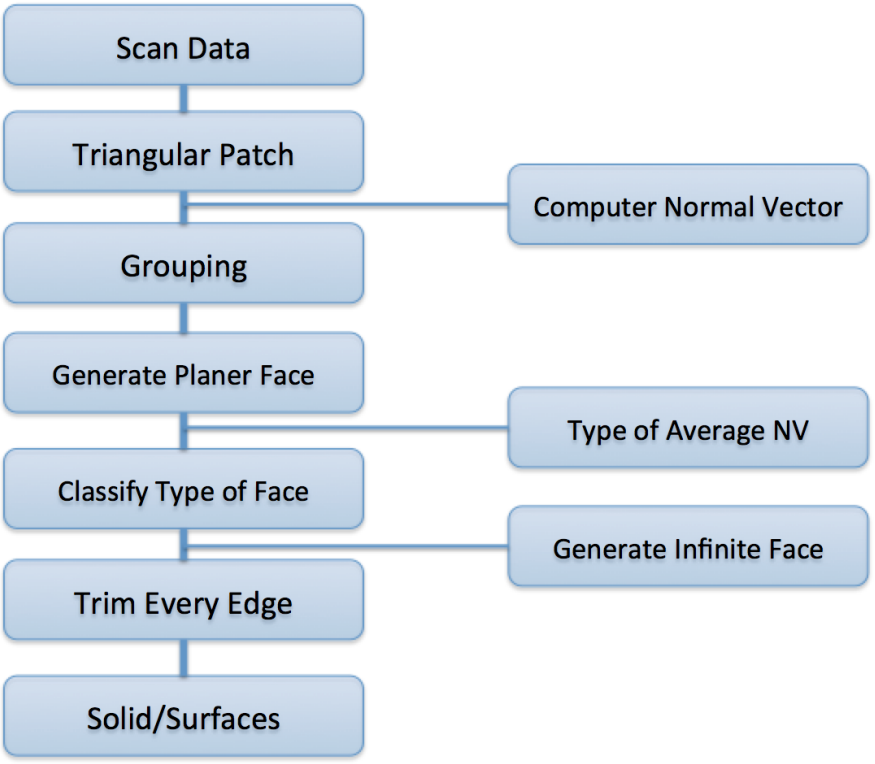


Figure 1.3: Overall procedure for the automatic solid data reconstruction. [2]

However, this method has several disadvantages, such as computational inefficiency because of classifying the points and segment. And it also cannot get the geometric parameter information from the process. Roseline [3] presented an automatic and comprehensive retro-engineering process dedicated mainly to 3D meshes obtained initially by mechanical object discretization. The process involves three steps. See Figure 1.4 However, this is just sample geometric primitives revolution and reconstruction. It cannot generate the complex models, such as airplane in one step and output the geometry feature that is important in aircraft design.

- Primitive extraction: in this step, the idea is first to detect the type of geometric primitive (i.e. a plane, sphere, cylinder or cone) that corresponds locally to the 3D mesh and to then compute the parameters which give the best fit. The method is based on differential geometry operators that characterize the local 3D shape.

- Wire construction: this is a key complex problem. It defines the relationship between all the extracted geometric primitives, which is subsequently used to compute intersection curves between two geometric primitives. Then all of these curves are combined to build a continuous wire in a consistent way.

- B-Rep creation: the B-Rep construction is presented. It consists of combining the information extracted or reconstructed during the two previous steps to construct a consistent model.

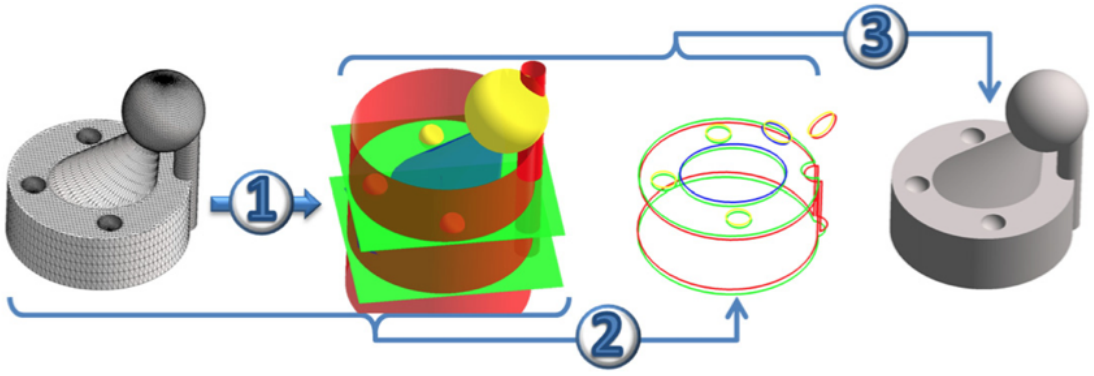


Figure 1.4: Overview of the reverse engineering method: Step 1: primitive extraction, Step 2: wire construction and Step 3: B-Rep creation. [3]

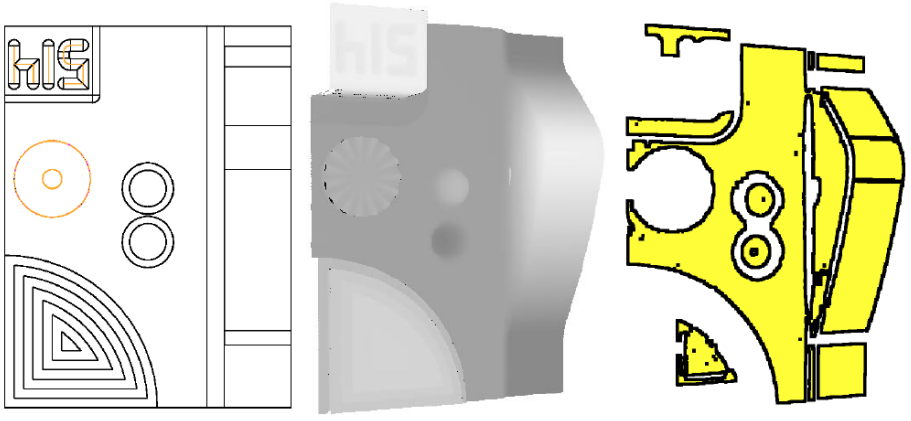
Huang and Menq [32] developed a systematic scheme and novel technologies to automatically reconstruct a CAD model from a set of points in a cloud scanned from the boundary surface of an existing object. The proposed scheme is composed of three major steps.

- Multiple input point clouds are incrementally integrated into a watertight triangle mesh to recover the object shape.

- Mesh segmentation is applied to the triangle mesh to extract individual geometric feature surfaces.
- The manifold topology describing the connectivity information between different geometric surfaces is automatically extracted and the mathematical description of each geometric feature is computed.

The computed topology and geometry information represented in the ACIS modeling kernel form a CAD model that may be used for various downstream applications. Compared with prior work, the proposed approach has the advantage that the processes of recognizing geometric features and of reconstructing CAD models are fully automated.

The object recognition in complex real environments in the presence of occlusion and clutter is a challenging task in reverse engineering. Bohm and Brenner [4] presented an object recognition process which is based on a CAD model of the object. Curvature information derived from the CAD model is used to support the feature extraction process. Reliable estimates of surface curvature are obtained from range images using a least-squares surface fitting algorithm. Figure 1.5 shows the feature extraction result from this method.



(a) wireframe display of the CAD model

(b) range image of the part

(c) the extracted surface patches from sensor data

Figure 1.5: Feature extraction on self-created part. [4]

Syed and Mohammed [33] developed a novel local surface description technique for automatic three dimensional (3D) object recognition. Fehr and Beksi [34] introduced techniques from other fields, such as image processing, into 3D point cloud processing in order to improve rendering, classification, and recognition. Li and Dong [35] proposed a method for object recognition based on Region of Interest (ROI) and Optimal Bag of Words model.

Geometry generated by parametric model has to be fitted to this point data. [1] Key research areas that still need further work before general-purpose reverse engineering becomes widely available include: improving data capture and calibration, coping with noise, merging views, coping with gaps in the data, reliable segmentation, fair surface fitting, recognizing natural or human-intended structure of the geometry of the object, and finally ensuring that consistent models are built. [26]

Model generation from the point data is a crucial step in the reverse engineering process. All the research above is based on generating the geometry model by a collection of surfaces. There may be gaps between the surfaces in the fitting result, and the impact of design parameters to the features of the whole geometry model is missed. The technique developed in this paper is aimed to overcome these problems. The data points will be fitted by one parametric geometry model directly, as opposed to as several discrete surfaces.

1.2.2 CAD Data Exchange

CAD Data exchange is a general field that geometry generation has wildly used. As complex geometric models require more concurrent engineering and faster response to the market demand, more communication between different systems is required. Most

CAD/CAM systems for different design fields are usually running on different environments and require different levels of support. And different CAD/CAM systems are very often found using different formats to display their drawing files, implying that a drawing developed by a system can sometimes not be represented by another system. This kind of problem usually involves a huge amount of data, different formats, and proprietary platforms. So, regenerating models in different system is a crucial research area in CAD data exchanging.

In order to communicate between different CAD/CAM systems, Chao and Wang [27] proposed a framework to exchange data between different CAD/CAM users. This framework consists of four parts: client databases, an index server, a CAD data format translator, and a file sharing control module. The architecture can be either built on the Internet or on intranet of a company. The CAD data format translator is developed to translate files of different formats into the STEP format. Xiao and Zheng [5] comprehensively studied the STEP-compliant systems and summarized their frameworks and criteria. A real STEP-NC data flow between a STEP-compliant CAD/CAM system and a STEP-CNC system is developed. (Flowchart is shown as Figure 1.6).

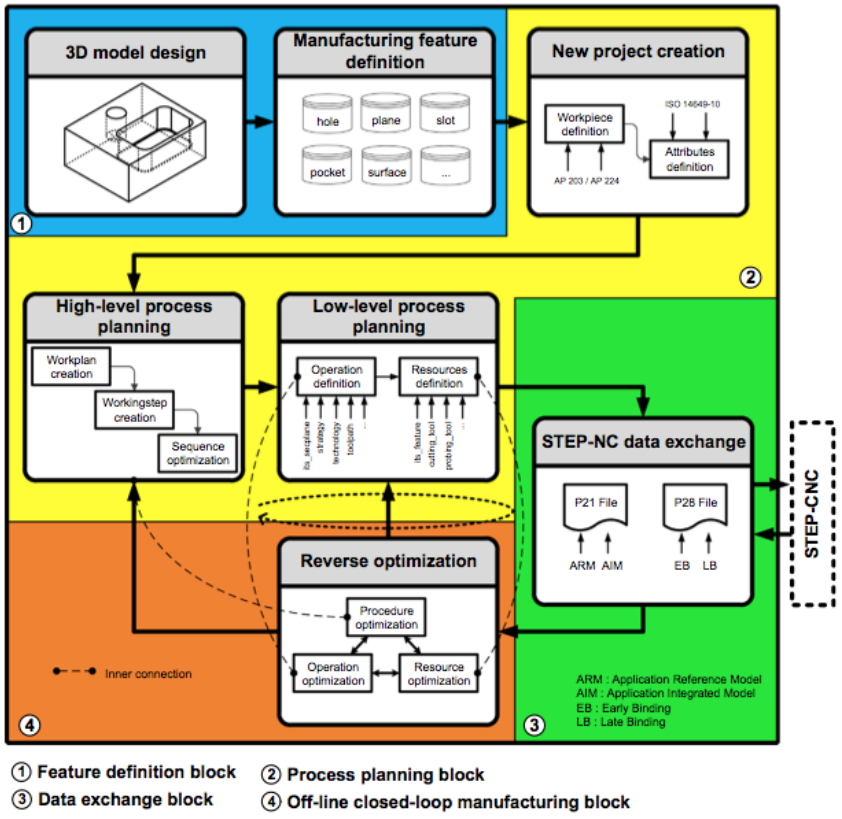


Figure 1.6: Framework of the proposed STEP-compliant CAD/CAM system.[5]

Kim and Pratt [36] suggested an implementational foundation for CAD data exchange with the preservation of design intent, based on the use of newly published parts of the International Standard ISO 10303 (STEP). Tsige-Tamirat and Fischer [37] described the features of the interface program McCad and present an application of the program to ITER torus sector model, which consists of all significant components. STEP-NC (Standard for the Exchange of Product model data) is the programming interface between CAD/CAM and CNC (computer numerical control) systems. Wu and Portheine [6] presented a software interface to automate the data exchange process between CAS and CAD/CAM systems with a voxel-based approach. This interface has to be implemented and integrated in the DISOS-system. The Standard Triangulation Language (STL) file format was chosen as the geometry data exchange format as in Figure 1.7.

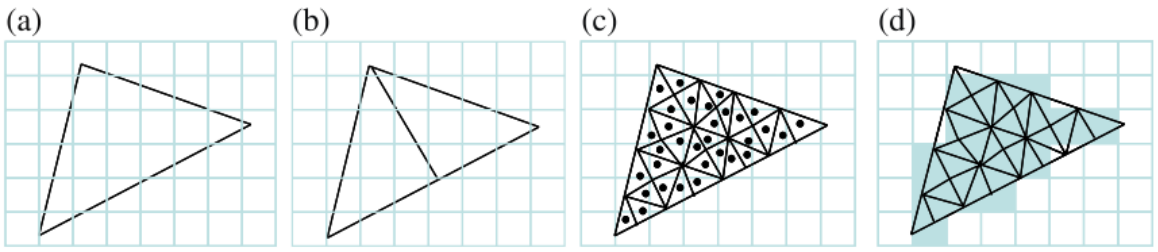


Figure 1.7: The subdivision of a triangle in 2D case.
(a) A triangle with floating coordinates at discretized grid of pixels.
(b) Subdividing triangle by splitting the longest edge.
(c) End of the subdivision when each triangle is smaller than the voxel size and the geometrical centers of each triangle.
(d) Labelling the pixel when it is within the object. [6]

CAD Data exchange also can be achieved by other methods. Mechanical CAD systems usually adopt a hybrid modeling approach in which both explicit and procedural models are utilized for the representation of 3D shapes. Procedural models are robust because they are not subject to the computational errors arising in the calculation of points, curves and surfaces that are characteristic of explicit (e.g., B-rep) models. Kim and Mun [38] described the concept of procedural 2D modeling as a method of representing procedural 2D CAD models in STEP in harmony with other STEP resources. The example of explicit and procedural models is shown in Figure 1.8. The board diagram on the left shows the state of the game at a specific time. However, such a diagram contains no information regarding how that state was arrived at. The overall history of the game is given by the sequence of moves, as shown on the righthand side of Figure 1.8. The state shown on the board diagram can be recovered by following this sequence.

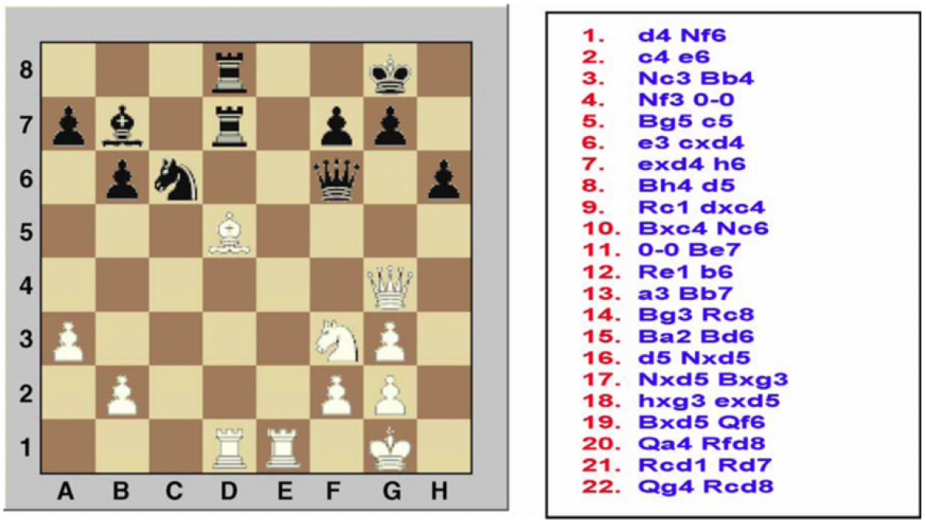


Figure 1.8: Explicit and procedural representations of a chess game [7]

Cheon and Kim [39] presented a new method to translate 3D data reconstructed from a free-hand 2D sketch into an editable form that reflects design intent so that the translated data can be directly used in 3D mechanical CAD systems. The feasibility of the proposed method has been demonstrated through experiments with prototype systems. Li and Kim [22] presented a method for resolving a problem that occurs from the exchange of design data between two different types of CAD systems. The problem stems from the difference in modeling units of the two systems. In this research, the conversion between two different modeling units is processed through direct and indirect mapping. Whyte and Bouchlaghem [40] described three different approaches to translate CAD data to virtual reality. The specific utilization of these approaches are in the 3D building design.

In this section, the researchers rely on diverse methods to exchange the data among different situations. The STEP method has a disadvantage that the original model's history needs to be provided during the representation process, such as feature tree of basic geometry models, Boolean operations and basic extrusion capabilities. However, this information always is lost during the redesign process. There is no research on how

to transfer data to complicated geometric parameters in one step without using pattern recognition to classify data into different basic parts, such circle, line sphere and so on. In this paper, these problems are solved by the new technique. The data points can be represented through generating parametric model once without classifying the data into different basic parts. It also do not need the history (feature tree) of the original geometry.

1.2.3 Parametric Model Reconstruction

Parametric model reconstruction is one of the most important problems in CAD, geometric modeling, computer graphics, and CAE. A Boundary Representation (B-Rep) [10] is often used to model objects in a computer. Parametric surface definitions, such as the Non Uniform Rational B-Spline (NURBS) surface, is one of the most commonly used B-Rep surface models to represent free-form objects due to its concise representation scheme.

Stamati and Antonopoulos [41] presented an approach to reconstructing geometry model and applied this approach into reconstructing traditional filigree jewelry. In this approach, the representation scheme for modeling filigree patterns uses elliptical arcs, Bezier segments, spirals and other curve segments. Ochmann and Vock [42] presented an automatic approach for the reconstruction of parametric 3D building models from indoor point clouds. Yoshihara and Yoshii [8] introduced a procedure for automatically reconstructing an arbitrary topological surface from an unorganized point data set; this surface had three representations, namely, quadrilateral meshes, Catmull-Clark subdivision surfaces, and B-spline surfaces (different representations are shown in Figure 1.9)

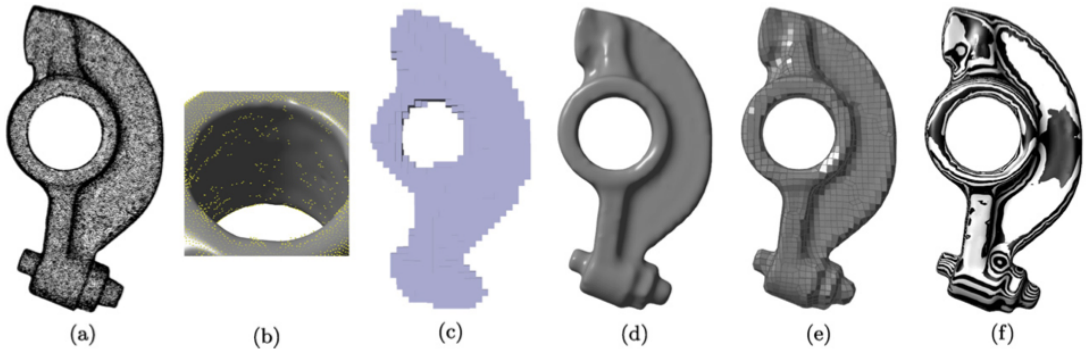


Figure 1.9: Rocker arm model: (a) Point clouds. (b) Sparse region. (c) Voxel model resulting from the level set method. (d) Resulting Catmull-Clark limit surface. (e) Resulting quadrilateral mesh. (f) Zebra mapping of the B-spline surface. [8]

Peethambaran and Muthuganapathy [43] introduced the concept of divergent concavity for simple, closed and planar curves and define a proximity graph called shape-hull graph which is capable of capturing the proximity of sample points. This approach for surface reconstruction is simple, non-parametric, single stage and reconstructs topologically correct piecewise linear approximation for divergent concave surfaces. Bauer and Polthier [9] present a complete process for parametric reconstruction of bent tube surfaces and propose a moving least squares method to compute the spine curve of a pipe surface from a point cloud of the surface. The algorithm for parametric reconstruction of bent tube surfaces performs the following steps, depicted in Figure 1.10

- First, the samples of the surface, shown in Fig 1.10 (b), are projected onto the spine curve. This procedure is described in Section 3. The result of this step is shown in Fig 1.10 (c).
- Next, reconstruct a polygonal curve from the spine point cloud using the NN-Crust algorithm of Dey and Kumar [10], as shown in Fig 1.10 (d).
- The polygonal curve is optionally simplified using Eu and Toussaint's algorithm to reduce the problem complexity for further computations.

- This polygonal curve is then approximated using an arc-line spline Fig 1.10 (e).
- Finally, the parametric description of the bent tube surface is optimized with regard to the least squares distance of the surface to the input samples using the Levenberg-Marquardt nonlinear optimization method. The final output reconstruction is shown in Fig 1.10 (f).

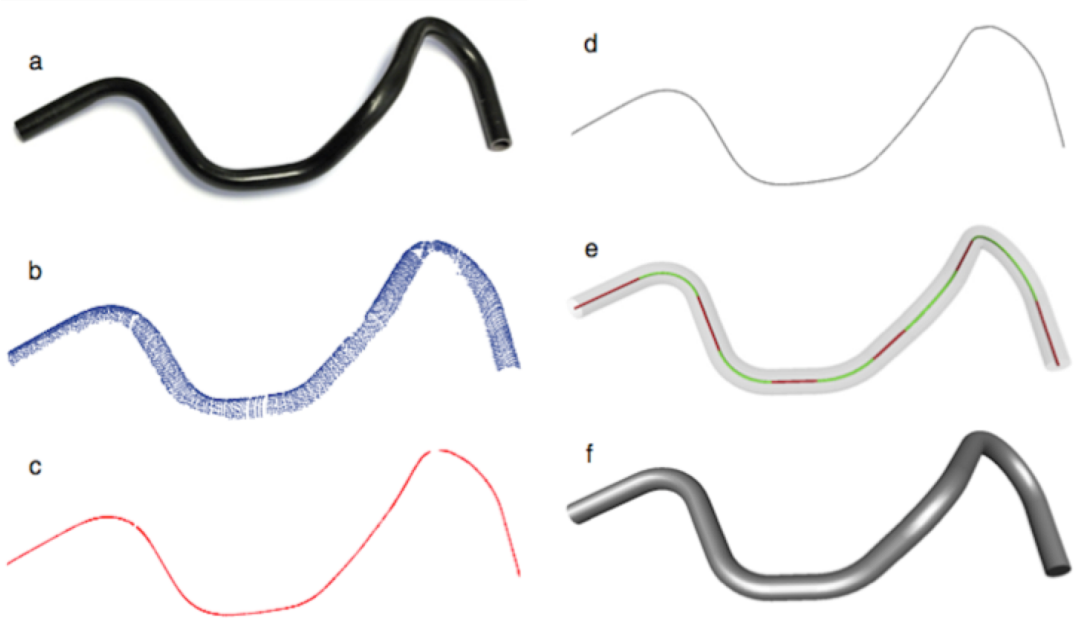


Figure 1.10: Reconstruction of a tube. (a) A photograph of the original tube. (b) A single- view, irregular scan of the tube surface. (c) Projection of the point cloud onto the MLS spine approximation. (d) Reconstruction of the spine curve. (e) Approximation of the spine curve by a G1 continuous arc-line spline. (f) The reconstructed tube surface. [9]

However, mesh models require significantly more storage. This results in large file size, creating an obstacle for model sharing between networked devices, particularly for wireless communication. So, there are other methods for reconstruction of the parametric models from a cloud of points or mesh. Lai and Yuen [10] proposed a blending scheme which is called a Hybrid PN Parametric Surface to blend a triangular mesh and a NURBS surface together. The blended surfaces are shown in Figure 1.11.

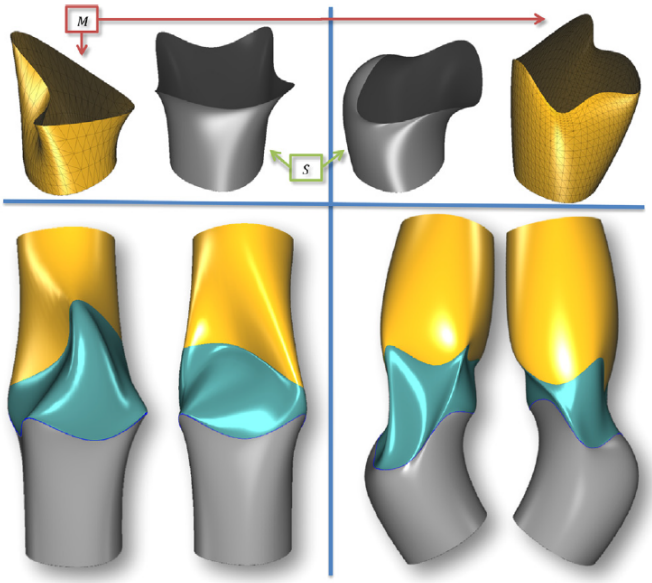


Figure 1.11: Open freeform meshes and NURBS surfaces with freeform boundaries. The upper part shows the inputs and lower part shows the front and back views of the results. The PN-Meshes are shaded in yellow. The input parametric surfaces are shaded in grey while the generated transition surfaces is shaded in cyan. [10]

In general, 3D scanned data points are noisy, contain outliers and holes, and have high variations in the point density. Therefore, all the research based on the free form surfaces (NURBS) is sensitive to the noisy data in the cloud of points. It is very easy to generate gaps between two surfaces and miss the geometry features. In this paper, the technique that generation of parametric geometry model (not segments) from a cloud of points at the same time can solve the noisy data problem.

1.2.4 Geometry Model Fitting

Model fitting can be classified as regular shape fitting and freeform parametric surface fitting. Both of them are fitting to the points in a cloud and generally are regarded as an optimization problem. Depending on the application, the conditions to be satisfied can

make the problem difficult to solve using classic methods, and for this reason, stochastic methods, such as genetic algorithms appear to be appropriate. [11, 23, 44]

Sevaux and Mineur [44] solved a curve fitting problem with the objective of generating shapes with specific curvature variations. The genetic algorithm was used as curve fitting method. Galvez and Iglesias [11] presented a novel hybrid evolutionary approach (called IMCH- GAPSO) for B-spline curve reconstruction comprised of two classical bio-inspired techniques: genetic algorithms (GA) and particle swarm optimization (PSO), accounting for data parameterization and knot placement, respectively. The flowchart is shown as Figure 1.12. The similar researches were done by Zhao, Deng *et al* [45, 46].

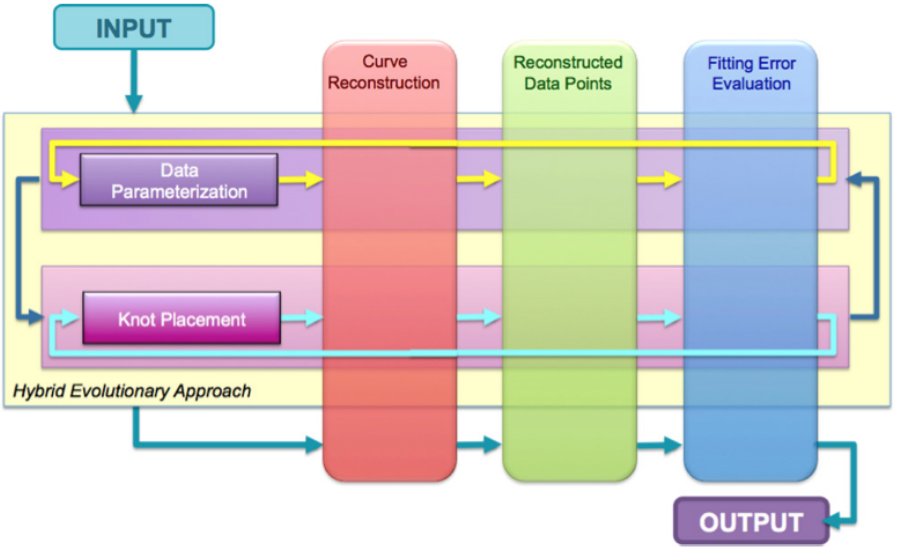


Figure 1.12: Graphical workflow of the proposed method. [11]

Lin [12] developed the adaptive data fitting algorithms by virtue of the local property of the Progressive-iterative approximation (PIA), which generates the fitting curve by adjusting the control points of a blending curve iteratively. In the adaptive data fitting algorithms, the control points are classified into two classes, namely, active and fixed control points. Only the active control points need to be adjusted in each iteration, thus saving computation greatly. The fitting process is shown in Figure 1.13. Kineri and

Wang [47] distinguished between two types of B-spline surface fitting — interpolation and approximation — and compare these two methods with standard fitting methods. Andrews and Sequin [48] introduced an approximate maximum-likelihood method to the kinematic surface fitting problem. Janunts [49] introduced a general straightforward and easy mathematical approach for modeling biconic surfaces and implicit parametric fitting to discrete corneal topographic height data.

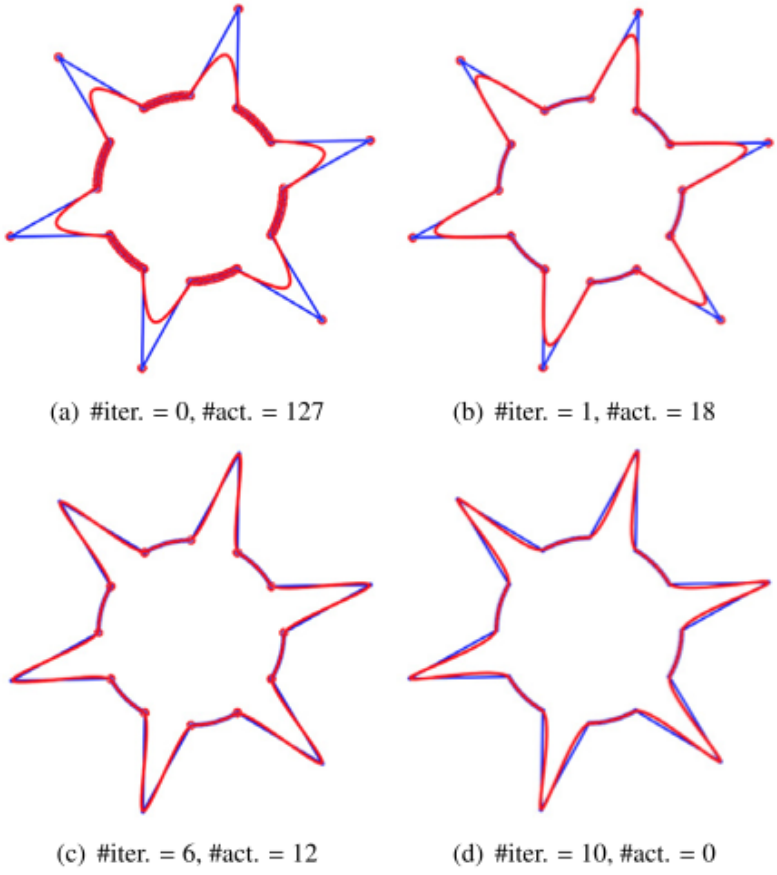


Figure 1.13: Fit the model star by the alternate method with a cubic B-spline curve (in red), where the data points not reaching the pre-defined fitting precision error = 10-2 are displayed in red. iter = the number of the iterations; act = the number of the active control points. [12]

Various methods have been proposed to calculate the fitting, e.g. computational geometry methods, support vector machines, and simplex methods. [50–52] But these techniques

have serious limitations and restricted applicability. They usually need to identify different cases and then appropriate manipulation is implemented according to the specific shape or point distribution. Kanatani [53] compared the convergence performance of different numerical schemes for geometric fitting and conclude that FNS exhibited the best convergence performance if initialized by Taubin's method. Therefore they are very inconvenient to be applied in practice. Gradient based optimization method is an efficient but not robust way for the geometry fitting problem. Flory, Zhang, Zheng and so on [13, 15, 54] applied gradient optimization method into B-spline curve fitting. The process is shown in Figure 1.14. More information will be discussed in the next section.

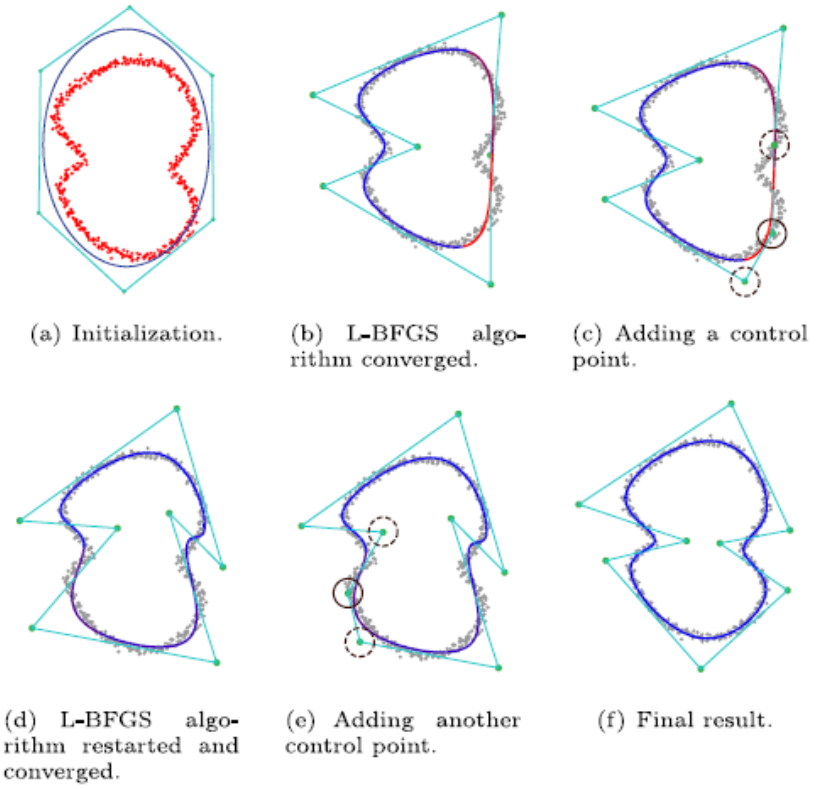


Figure 1.14: The L-BFGS fitting method process. [13]

Objective function selecting is also an important problem in geometry fitting. Bo and Ling [23] studied the performance of algorithms for freeform surface fitting when PD, SD and TD error terms are used as quadratic approximations to the squared orthogonal distances

from data points to the fitting surface. Based on their experimental results, using the TD error term and the SD error term leads to surface fitting algorithms that converge much faster than using the PD error term. Zhang and Gu [14] proposed a curve fitting approach called adaptive moving total least squares (AMTLS) method. This method uses distance between the given points and fitting curve as objective function, as shown in Figure 1.15. Similar research was done by Wang, Flory *et al* [55, 56]. Pourkarimi and Wang [57, 58] proposed a multiobjective linear programming model for the curve fitting problem. In this model, all of the violations of the corresponding polynomial fitted curve are minimized simultaneously as a vector.

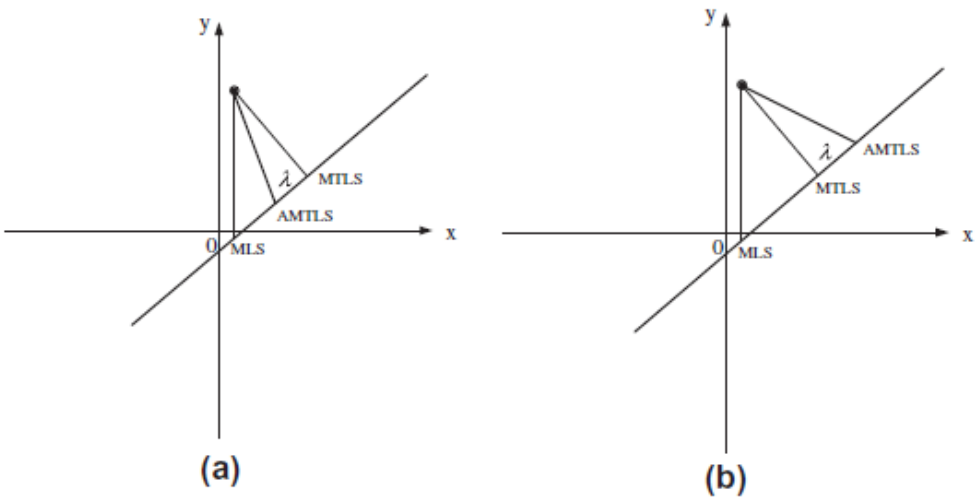


Figure 1.15: Local approximants of AMTLS. [14]

Instead of fitting geometry models to point of clouds, other researchers focus on fitting geometry models into closed regions. Chaudhuri and Samal [59] introduced a new approach for fitting of a bounding rectangle to closed regions. This is similar with triangles fitting approach which is introduced before section [6]. Zhang and He [15] presented a fast and powerful method to evaluate the Minimum Zone form errors (Figure 1.16) of general complex surfaces. The original minimum optimization is transferred into an unconstrained differentiable minimization problem by exponential penalty functions.

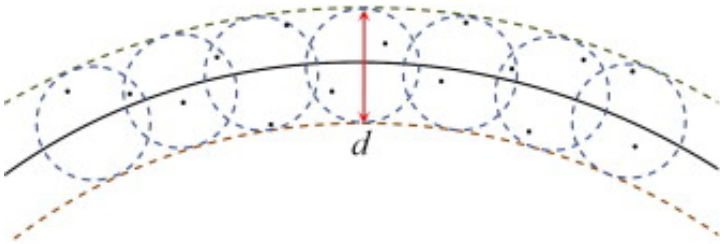


Figure 1.16: Minimum zone fitting of complex surfaces. [15]

Geometry fitting is a very important subject of research in fields such as geometric modeling and computer-aided design/ manufacturing (CAD/CAM) [28]. Optimization techniques are the main methods for solving this kind problem. Flory [16] described the fitting problem as an optimization problem and employs an iterative procedure to solve it. The presence of obstacles poses constraints on this minimization process. There are two families of obstacles: first, the point cloud itself is interpreted as obstacle, such as minimum zone form error. Second, arbitrary regions is defined as the fitting must not penetrate. (Figure 1.17) Wang [60] also introduced an approach to fit a geometric shape to image.

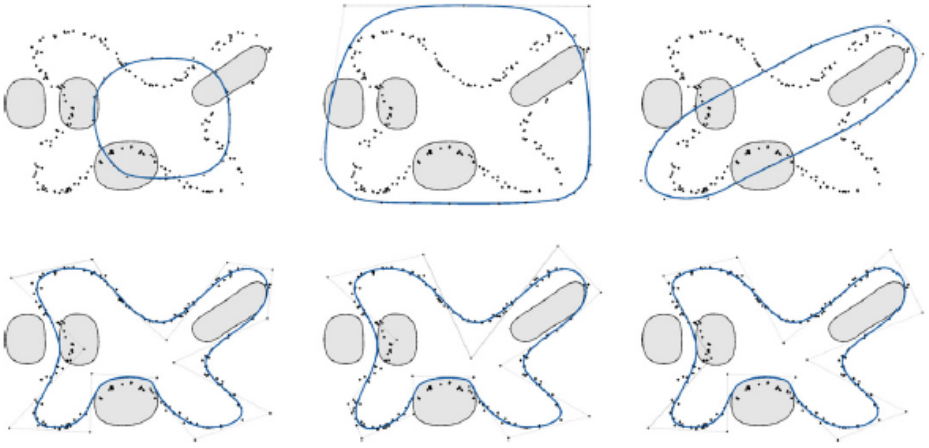


Figure 1.17: Fitting a cubic B-spline curve to a point cloud in the presence of four general obstacles. For three different initial setups (top row), the corresponding final approximations after 25 iterations (bottom row) are shown. [16]

Galvez [61] presented a new approach for data fitting with B-spline curve. This scheme is based on the idea of considering the internal knots as free variables of the problem, which leads to a very difficult continuous multimodal and multivariate nonlinear optimization problem. To solve this problem, particle swarm optimization (PSO) paradigm is applied to compute an optimal knot vector automatically. The similar researches were done by Galvez [62]. Kang and Chen [63] introduced a framework for computing knots in curve fitting based on a sparse optimization model. This framework consists of two steps: first, from a dense initial knot vector, a set of active knots is selected at which certain order derivative of the spline is discontinuous by solving a sparse optimization problem; second, remove redundant knots and adjust the positions of active knots to obtain the final knot vector. Chou and Cheng [17] proposed an optimized hybrid artificial intelligence model to integrate a fast messy genetic algorithm (fmGA) with a support vector machine (SVM). The flowchart is shown in Figure 1.18.

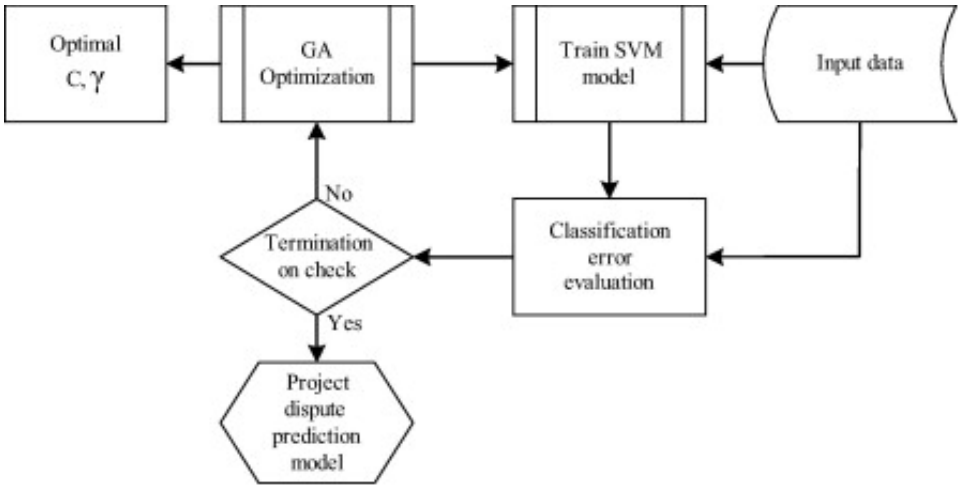


Figure 1.18: GA-based SVM flowchart. [17]

Optimum design is another application field of optimization method in geometry model fitting. The problem of efficient shape parameterizations is crucial to optimization of geometry and shape that has increasingly come under consideration in practice. Carrizosa

and Morales [64] emphasized some links between mathematical optimization methods and supervised classification. Liu and Shimoda [65] proposed a parameter-free shape optimization method for designing the shapes of stiffened thin-walled or shell structures in the natural vibration problem. The optimal free boundary shapes of either stiffeners or basic structures can be obtained with this method. [18] General optimum design process is shown in Figure 1.19

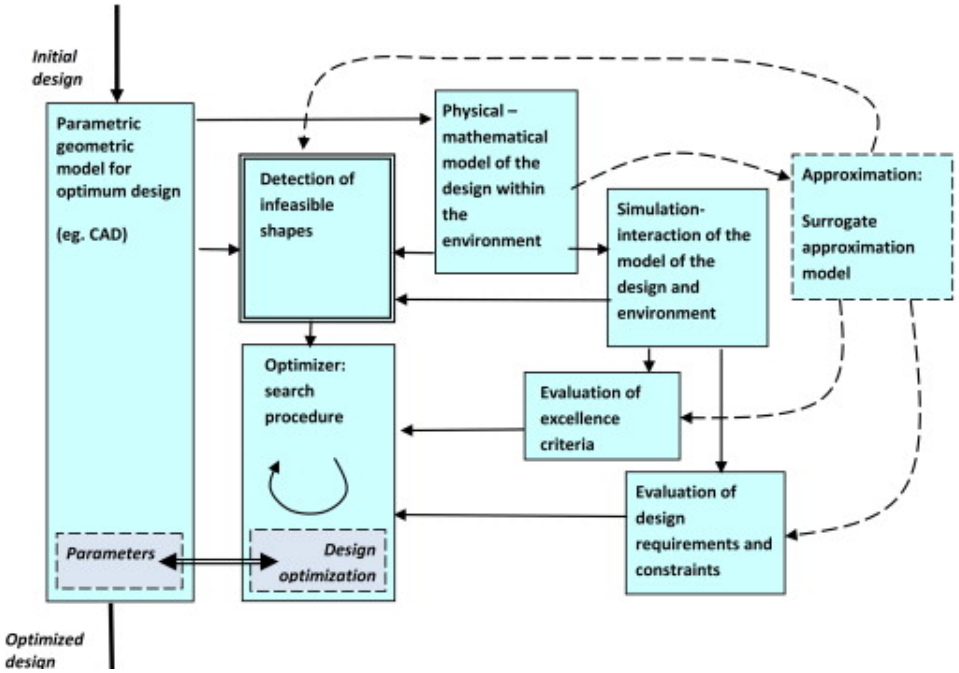


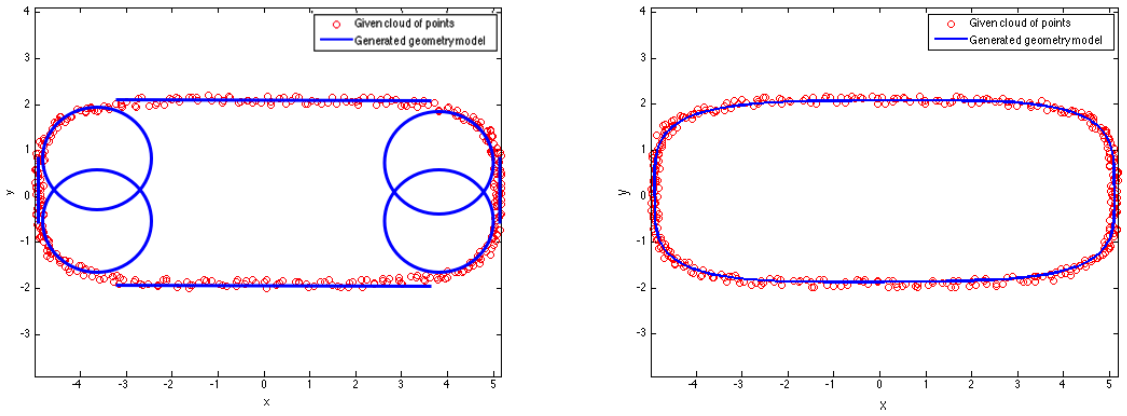
Figure 1.19: Basic components of the computer-aided optimum design process. [18]

Previous researchers have focused on a specific step of the geometry fitting problem, either data parameterization (fixed-knots methods) or knot placement (variable knots methods). So far, no evolutionary approach has been successfully applied to the general complex parametric geometry fitting problem that the fitting process will consider the whole data at the same time but not break them into pieces. The propose of the work in this paper is developing the approach that can achieve the goal above.

To sum up, all the geometry generation problem introduced in the above four sections are solved based on traditional method. In that method, the cloud of points is classified into several parts and fitted by discrete surfaces/curves (NURBS). It has several disadvantages:

- The design parameters of discrete surfaces/curves can not present the geometry features of the whole model
- The method is very sensitive to noise in the data. The fitting results of surfaces always have gets between each other.
- Because the geometry model is formed by a bunch of discrete surfaces, the heuristic optimization is the preferred technique during solving this kind problem. But the heuristic method is not as efficient as gradient-based optimization method in generally.

Therefore, a new technique is developed in this parer for solving the problems listed above. In the new technique, the fitting process uses a gradient optimization technique applied to the whole cloud, without the need to segment or classify the cloud points for different faces. The basic elements in geometry model are bodies but not surfaces or curves. The comparison between traditional and new techniques is shown as below:



(a) Traditional fitting technique

(b) New fitting technique

Figure 1.20: Optimization problem

Figure 1.20a shows the traditional free-form fitting technique. The cloud of points is fitted by 4 straight lines and circles. As shown in the figure, there are gaps between lines and circles due to the noisy points in the cloud. Figure 1.20b shows the new technique developed in this paper for fitting the cloud of points through integral parametric geometry model. The cloud of points is fitted by a super-ellipse model. There are only 5 design parameters (2 radii, 1 rotation angle, x,y coordinates of center and power) to control the geometry shape. In this method, there are no "gaps" problem due to the points are fitted at the same time. And the fitting problem can be easily expressed mathematically for applying the gradient-based optimization.

1.2.5 Aircraft Parameters Analysis

Aircraft are geometrically complicated models that contains many parameters, such as wing area, aspect ratio, span. The problem regarding aircraft parameters analysis can be classified into two fields. One is the identification of aircraft [29, 66–69]. The other

is the design of aircraft's. My work is more regarding the second part. The basic idea is generating a parametric aircraft model from a cloud of points first, then one can analysis the affect of design parameters to the aircraft geometry shape.

In the analysis and design of aircraft, NURBS and subdivision surfaces are the important design tools. When using polynomial based methods [19] the surface patch is usually generated using a set of control points, by manipulating these control points the surface shape changes. One problem that arises here is that there are often too many control points to manipulate the underlying geometry. Thus, there are much research to develop techniques that allow the user to manipulate the surface effectively using minimum number of controls.

Augsdorfer [70] presented methods for analyzing the magnitude of artifacts in a surface defined by a quadrilateral control mesh and uses the subdivision process as a tool for analysis. The results provided a measure of surface artifacts with respect to initial control point sampling for all B-Splines, quadrilateral box-spline surfaces and regular regions of subdivision surfaces. Athanasopoulos and Ugail [19] present a surface generation tool designed for the construction of aircraft geometry and the surface generation is based on Partial Differential Equations (PDEs). The step of generate aircraft is shown in Figure 1.21.

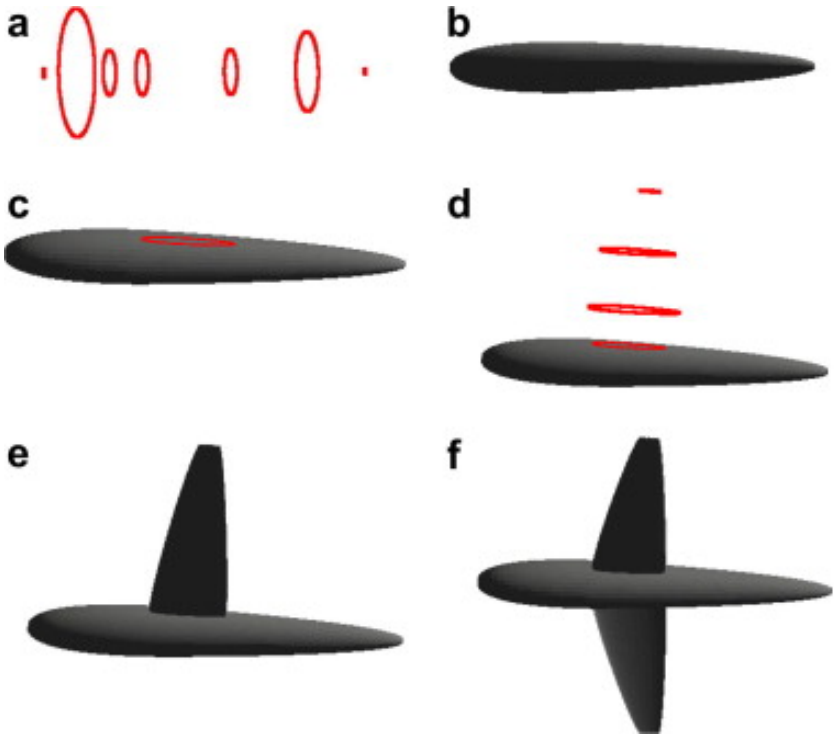


Figure 1.21: Construction of a generic aircraft shape. (a) Fuselage curves. (b) Two patches fuselage object. (c) The initial curve for the wing. (d) The generating curves of the wing. (e) Wing and fuselage objects blended. (f) The final basic shape of the airplane. [19]

One of the major tasks in the design phase of a new aircraft is the definition of its configuration along with the main geometric characteristics. For an aerodynamic early phase conceptual design, a step before the application of CAD is needed, i.e. a toolbox that will produce generic and parameterized aerodynamic surfaces, which will take into account the special needs and constraints for the conceptual design of an aircraft. Besides the well known general CAD packages, very few are specialized in aircraft design. Byrne and Cardiff [20] combine NASA's parametric aircraft system (OpenVSP) and a computational fluid dynamics solver (OpenFOAM) with an evolutionary algorithm to generate a variety of optimized and novel designs.

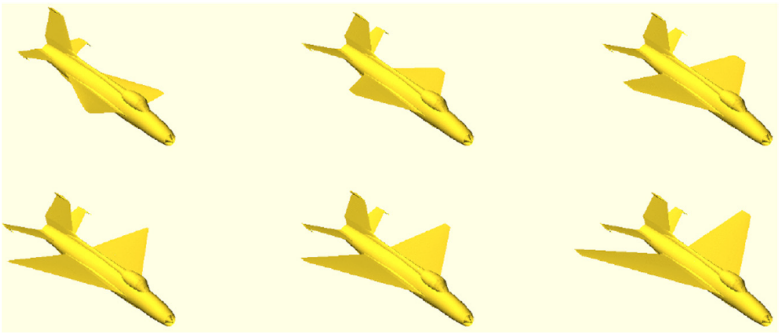


Figure 1.22: Selected individuals from the pareto front of the MIG 21 optimization results in the order of increasing lift (drag) from top left to bottom right. [20]

Ledermann and Hanske [21] illustrate how beneficial parametric-associative CAE methods are in aircraft pre-design. The knowledge based geometry can serve as a basis for different domains like structural analysis, computational fluid dynamics, and others. Different kinds aircraft models are shown in Figure 1.23.

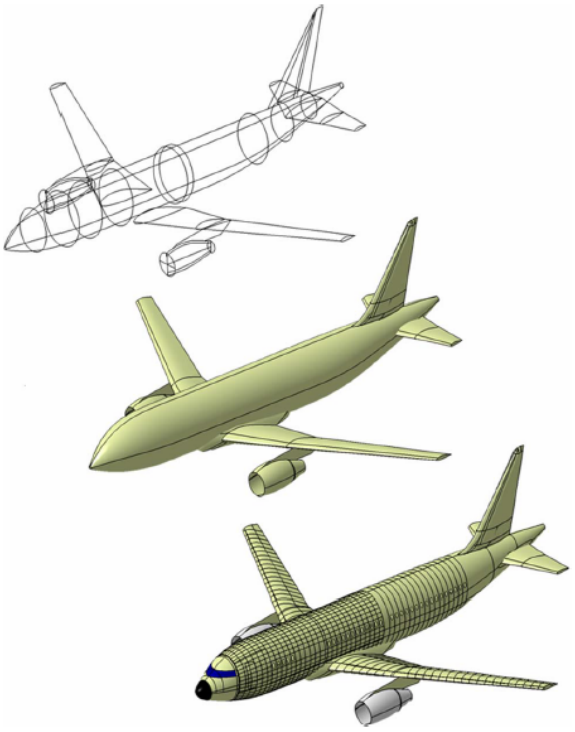


Figure 1.23: Geometric aircraft model with different levels of detail: (top) wireframe, (middle) master geometry, (bottom) including some inner geometry. [21]

For the aircraft parameters analysis problem, an efficient, robust and accurate method for generating parametric aircraft models from a cloud of point is necessary. In my research, using an optimization method applied to the whole data at the same time, but does not segment or classify the cloud points into different type pieces. This will significantly improve the efficiency of algorithm.

Chapter 2

Optimization Techniques

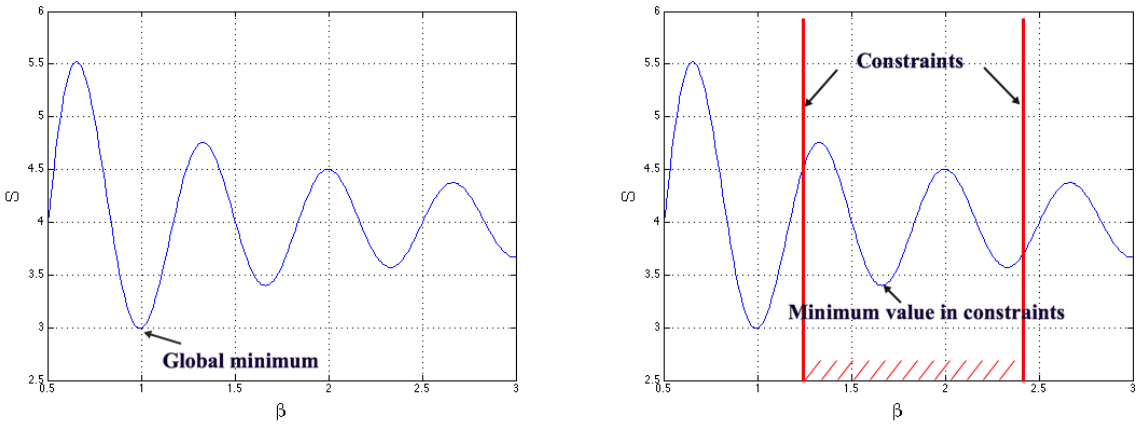
2.1 Introduction

Now that the idea of parametric model generating and its applications has been discussed, optimization technique is the general method for solving this kind problems. In this section, the standard optimization techniques are discussed. These include gradient based optimization method and heuristics optimization methods.

2.1.1 General Format for Optimization Problem

An optimization technique is the selection of a best candidate solution (with regard to some criterion) from some set of available alternatives.[71] In general, an optimization algorithm is a method that finds maximum or minimum of an objective function by changing input parameters from within an allowed set. Figure 2.1 shows two types optimization problems. Figure 2.1a shows the unconstrained optimization problem. In this kind of problem, the input parameters can be changed in global area to minimize or

maximize the objective function. On the other hand, Figure 2.1b shows the constrained optimization problem that the input parameters only can be changed in a constrained area to minimize or maximize the objective function.



(a) Unconstrained (b) Constrained

Figure 2.1: Example of optimization problem

For applying optimization techniques, the objective function needs to be defined at first. The standard form for a multi-objective, non-linear, constrained optimization problem is provided in Equation 2.1a below

$$\text{minimize :} \quad S_{ic}(\vec{\beta}) \qquad ic = 1 \ldots m_c \tag{2.1a}$$

$$\text{subject to :} \quad C1_{jc}(\vec{\beta}) <= 0 \qquad jc = 1 \ldots n_c \tag{2.1b}$$

$$C2_{kc}(\vec{\beta}) = 0 \qquad kc = 1 \ldots p_c \tag{2.1c}$$

$$\beta_{lcL} <= \beta_{lc} <= \beta_{lcU} \qquad lc = 1 \ldots q_c \tag{2.1d}$$

In Equation 2.1a, $S_{ic}(\vec{\beta})$ represents the multiple objective functions with vector parameter $\vec{\beta}$. Solving a multi-objective optimization problem is sometimes understood as approximating or computing all or a representative set of Pareto optimal solutions. The solution

is called Pareto optimal, if none of the objective functions can be improved in value without degrading some of the other objective values. [72, 73]

Scalarizing a multi-objective optimization problem is an a priori method, which means formulating a single-objective optimization problem such that optimal solutions to the single-objective optimization problem are Pareto optimal solutions to the multi-objective optimization problem.[74] If scalarization is done carefully, Pareto optimality of the solutions obtained can be guaranteed. The most general scalarization of multi-objective optimization problems is linear scalarization (Equation 2.2).

$$S_{new} = \min_{\beta_{lcL} <= \beta_{lc} <= \beta_{lcU}} \sum_{ic=1}^{mc} w_{ic} \cdot S_{ic}(\vec{\beta}) \tag{2.2}$$

where the weights of the objectives $w_{ic} > 0$ are the parameters of the scalarization.

In Equation 2.1b, $C1_{jc}(\vec{\beta})$ is an inequality constraint and $C2_{kc}(\vec{\beta})$ is an equality constraint function. The vector $\vec{\beta}$ represents the q_c design variables that are modified to obtain the optimum of objective function. The ranges of design parameters are defined by the upper and lower bounds of the design variables β_{lcL} and β_{lcU}. In the general case, the objective and constraint functions can be linear or non-linear functions. [75].

For the generation of parametric model from a cloud of points, the problem is mathematized as single objective function and un-constraint optimization problem. Therefore, in this thesis, the solution for this kind problem is concerned. For minimizing/maximizing the single un-constraint objective function, the gradient-based [76] and heuristics [77] optimization algorithms will be introduced in the following section.

2.1.2 Test Function for Optimization Technique

For comparing the different optimization algorithm, Modified Styblinski-Tang function is used as the test function. The original Styblinski-Tang function is shown as Equation 2.3 and its 3D plot is shown in Figure 2.2.

$$S(\vec{\beta}) = \frac{\sum_{i=1}^{2} \beta_i^4 - 16\beta_i^2 + 5\beta_i}{2} \tag{2.3}$$

where, $\vec{\beta}$ is the parameters need to be optimized. The global minimum is $\vec{\beta} = [-2.903534, -2.903534]$

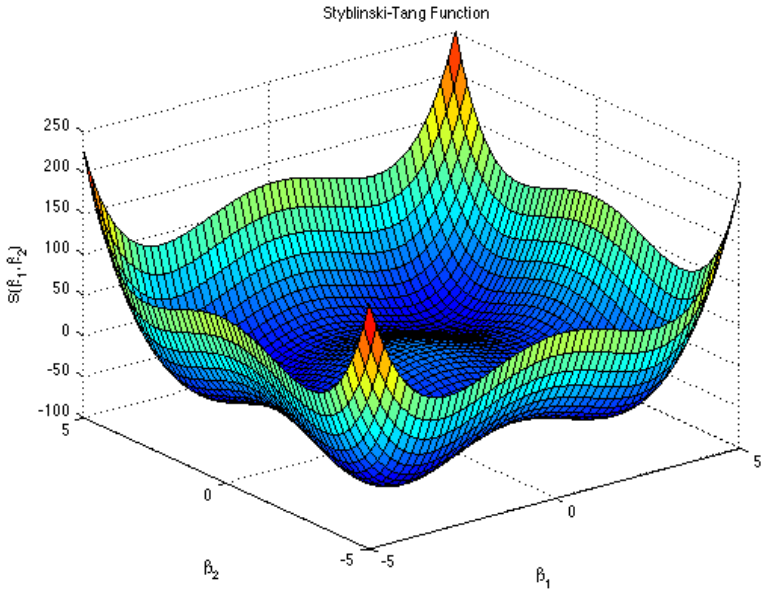


Figure 2.2: 3D plot of Styblinski-Tang function

Because the Levenberg-Marquardt method (will be introduced in Section 2.3.3) will take the structural advantage of least square problem, the original Styblinski-Tang need be squared. Moreover, for keeping the shape of Styblinski-Tang function (3 local minimum and 1 global minimum), a constant (choose 80 here) is need to add to the original Styblinski-Tang. The Modified Styblinski-Tang function is shown as Equation 2.4 and

its 3D plot is shown as Figure 2.3 As shown in the figure, there are 4 minimum values in the Styblinski-Tang function and the global minimum value is one of them. Therefore, Modified Styblinski-Tang (MST) function is a good choice for testing the performance of optimization algorithm (Gradient decent, Newton's, Levenberg-Marquardt, Improved Levenberg-Marquardt).

$$S(\vec{\beta}) = (\frac{\sum_{i=1}^{2} \beta_i^4 - 16\beta_i^2 + 5\beta_i}{2} + 80)^2 \tag{2.4}$$

where, $\vec{\beta}$ is the parameters need to be optimized. The global minimum is same as the original Styblinski-Tang function

$\vec{\beta} = [-2.903534, -2.903534]$

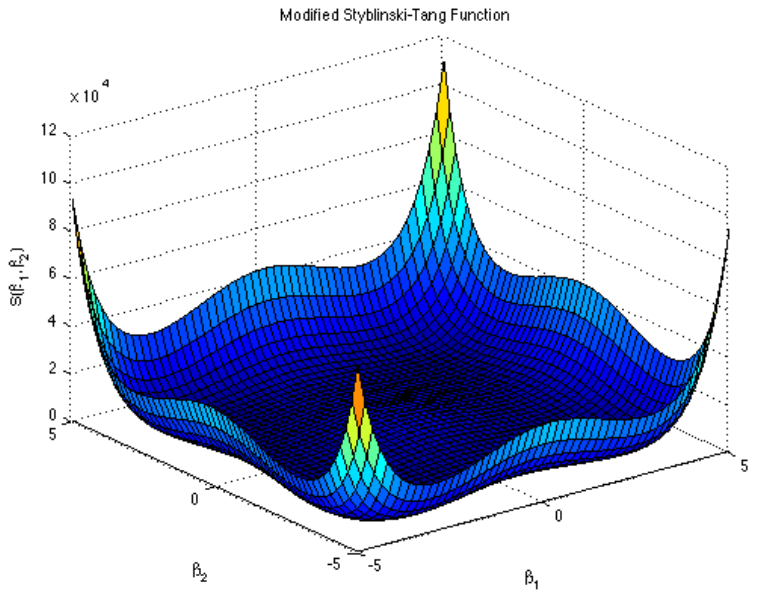


Figure 2.3: 3D plot of modified Styblinski-Tang function (MST)

2.2 Heuristics Methods

The general optimization method for solving the global minimum problem is heuristics optimization method. A heuristic method is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. For example, it may approximate the exact solution. [77–80] However, it is not guaranteed mathematically to find the solution. For demonstrating how heuristics optimization method works, simulated annealing (SA) method is applied. The reason of choosing SA is that the basic idea of SA (accept not too bad results) will be borrowed into gradient based method for solving the generation parametric model problem later.

2.2.1 Simulated Annealing algorithm

Simulated annealing (SA) [81] was originally motivated by the process of physical annealing in metal work and successfully used in optimization. SA uses a Metropolis criterion [82] to have a better chance to obtain the global minimum and escape from being trapped in a local minimum energy state. It interprets slow cooling as a slow decrease in the probability of accepting worse solutions as it explores the solution space. Accepting worse solutions is a fundamental property of heuristics because it allows for a more extensive search for the optimal solution.

In SA algorithm, there are several parameters need to be clarified. The energy state E is the same as the objective function. The parameter settings of the SA algorithm include: the initial temperature T_0, the freezing temperature T_f, the design variables X_0, the energy state of X_0 as $E(X_0)$, and the temperature decrement factor K.

The SA generates a random perturbation that displaces a particle [83] (moving the configuration of X_0 to the other configuration, X_i). If the new configuration has a lower

energy state, the move is accepted ($X_0 = X_i$, $E(X_i) = E(X_0)$). Otherwise, the move is accepted, with the acceptance probability of the Metropolis criterion.

For any given finite problem, the probability that the simulated annealing algorithm terminates with a global optimal solution approaches 1 as the annealing schedule is extended. This theoretical result, however, is not particularly helpful, since the time required to ensure a significant probability of success will usually exceed the time required for a complete search of the solution space. [84]

The pseudocode of Simulated Annealing algorithm is shown as below:

```
Let X = X0
For i = 1 through imax
        Pick a random neighbour, Xnew = neighbour(X)
        If P(E(X), E(Xnew), T) > random(0, 1)
        X = Xnew
Output: the final state X
```

The probability function of P is as Equation 2.5.

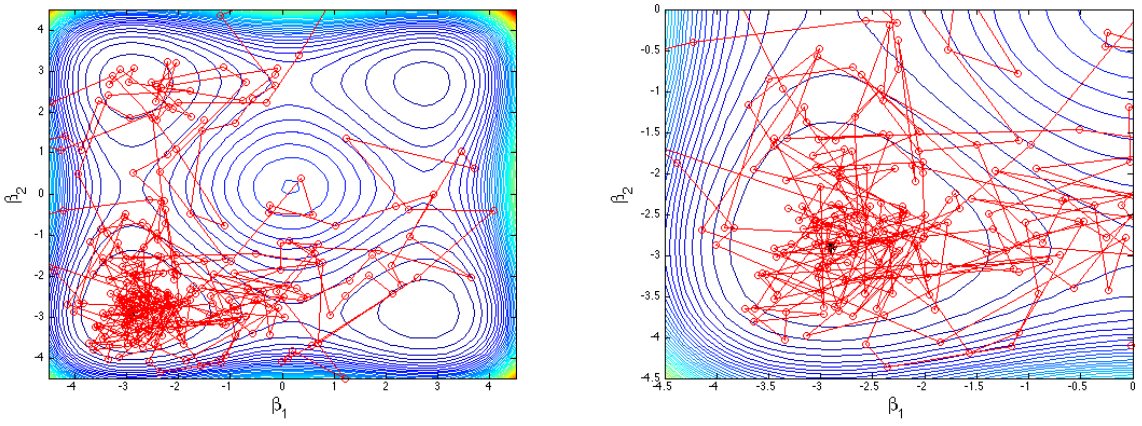
$$P = exp(-\frac{E(X_i) - E(X_0)}{KT}) = exp(-\frac{\Delta E}{T}) \tag{2.5}$$

Applying the SA method into optimizing the modified Styblinski-Tang function, the probability function 2.5 becomes Equation 2.6:

$$P = exp(-\frac{S(\vec{\beta}^{(i+1)}) - S(\vec{\beta}^{(i)})}{KT}) = exp(-\frac{\Delta S}{T}) \tag{2.6}$$

After applying the SA method, the optimization iterations is shown in the Figure 2.4. The red lines are the trajectory of optimization process and the blue star is the global

minimum of the objective function. For the simulated annealing method, the optimization process stopped at the global minimum value and the parameter $\vec{\beta}$ is very close to the correct value. However, there are too many iterations during the optimization process. Because only the objective function is calculated during each iteration, the number of function evaluations is same as the iteration number for the SA method. In this specific case, the function evaluation is 825.



(a) Trajectory of convergence

(b) Zoom in the figure

Figure 2.4: Trajectory of objective function value based on simulated annealing

The log of objective function value after each iteration is shown in Figure 2.5. The objective value has large fluctuations at the beginning, and the fluctuation value is reduced during the optimization process. This is the basic idea of SA method that accepted some "uphill" values at the beginning. At the end, only "downhill" steps are accepted. This idea will be borrowed into gradient based optimization method in later section.

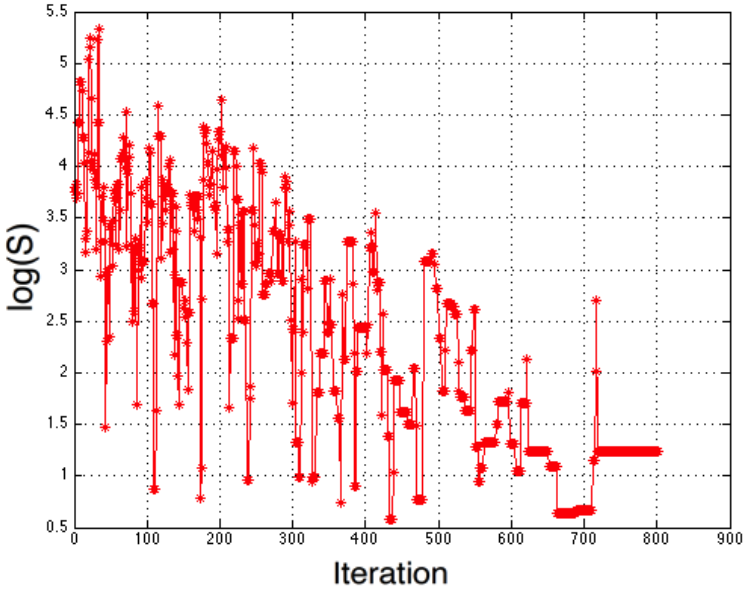


Figure 2.5: Log of objective function value at each iteration based on simulated annealing

Although, the SA method can find the global minimum value of the objective function, there is no guarantee that will be achieved every time. This means the SA algorithm (heuristic method) cannot get the exactly same results when run the same problem twice. Moreover, there are also a lot of function evaluations during the optimization process when using the SA method. Therefore, gradient based optimization algorithms need to be considered for improved the robustness and accuracy of the optimization.

2.3 Gradient Based Methods

Gradient-based optimizations [85–87] that use derivatives information are widely used in many kinds of problems, such as: mechanics [88–93], economics [94–97] , control engineering [98–103] , traffic assignment [104] problem and so on. As indicated by the name,

gradient-based optimization techniques make use of gradient information to find the minimum/maximum solution of the objective function. The benefit of these techniques is that search direction [105, 106] information provided by the gradient.

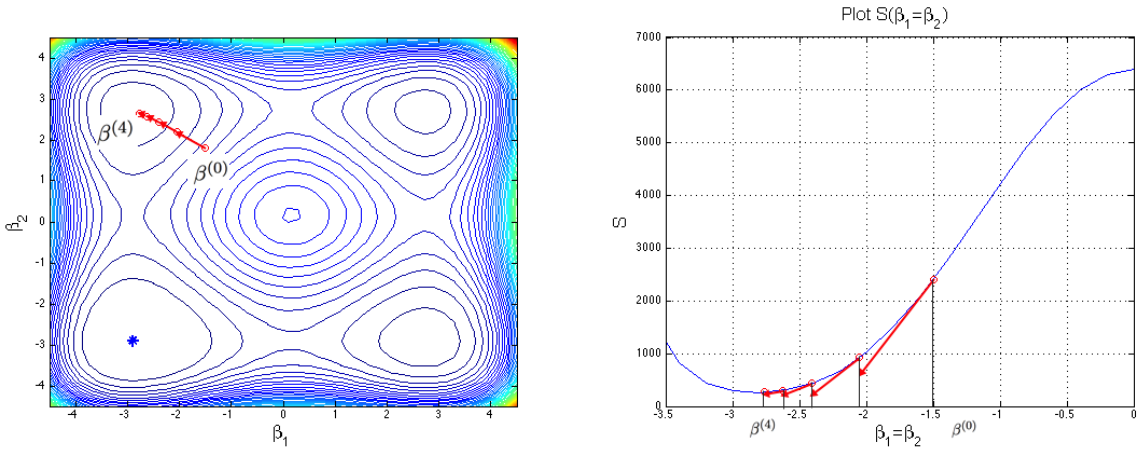
2.3.1 Gradient Decent Method

Gradient descent is a first-order iterative optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. This can be expressed as Equation 2.7

$$\vec{\beta}^{(s+1)} = \vec{\beta}^{(s)} - \gamma S'(\vec{\beta}^{(s)}) \tag{2.7}$$

where $\vec{\beta}$ is the design variable, (s) is the number of iterations, γ is the step size as a constant number. For making sure the optimizer does not pass over the minimum value, γ should be set as a small number. S' is the first derivative of objective function.

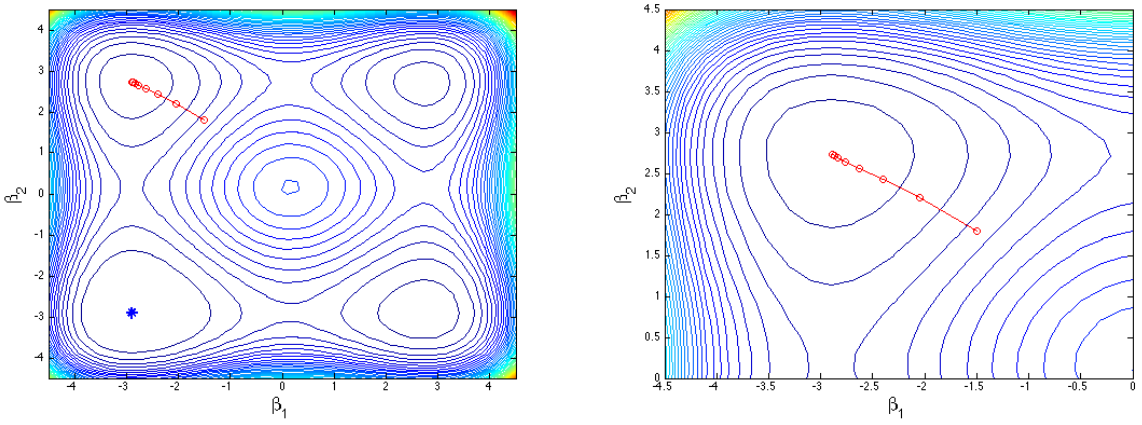
Figure 2.6 shows several iterations after applying the gradient decent method (using first derivative information for providing the search direction) for optimizing modified Styblinski-Tang function. In the Figure 2.6a, $\beta^{(0)}...\beta^{(4)}$ are the result parameters of each search step. Red arrows are the search direction for each step. Because the search direction is decided by the gradient information, the search direction at each step is perpendicular to the isogradient line at that step. The total step size is getting smaller during the optimization process because the gradient value is reduced to zero at the minimum value.The section along the optimizer moving direction is shown in Figure 2.6b. The blue line is the section curve of the modified Styblinski-Tang function, and the red lines are the iteration steps.



(a) Contour of the modified Styblinski-Tang (b) Section curve of the MST

Figure 2.6: Search directions provided by gradient decent (first derivative information)

Applying the gradient decent method to optimizing MST, in the Figure 2.7, the red lines are the trajectory of optimization process and the blue star is the global minimum of the objective function. For the gradient decent method, the optimization process stopped at local minimum on the up left corner, because the steps only move to the gradient decent directions. The number of iterations is reduced a lot comparing with the SA method. There are only 7 iterations till converge. Because the first derivatives information for $\vec{\beta}$ (2 design parameters) need to be calculated in each iteration, there are total 3 function evaluations in each iteration. The total function evaluations for gradient decent method is 21.



(a) Trajectory of convergence (b) Zoom in the figure

Figure 2.7: Trajectory of objective function value based on gradient decent method

The log of objective function value after each iteration is shown in Figure 2.8.

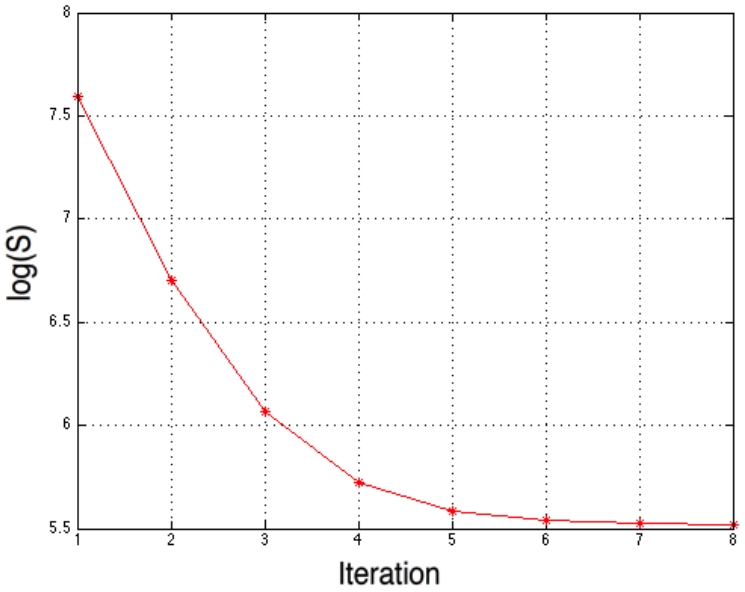


Figure 2.8: Log of objective function value at each iteration based on gradient decent

Gradient decent method reduced the function evolution a lot, but it is difficult for choosing a good step size γ. When the iteration is around the minimum value, the total step

size $\gamma S'(\vec{\beta})$ will be very small due to the $S' \approx 0$. Therefore, Newton's method will be introduced for overcoming this problem.

2.3.2 Newton's Method

Original Newton's method is used for finding roots (zero points) of a function S. The step size of each iteration is determined by intersection points between tangent lines and $S = 0$ line. Therefore, Newton's method can optimize the step size automatically in each iteration. For the optimization problem, the Newton's method can be used as finding the roots of first derivative of S ($S' = 0$). The mathematical explanation is as below:

Newton's algorithm is an unconstrained, second order derivative optimization algorithm [107–112]. that is derived from a second-order Taylor series expansion of the objective function. The second order Taylor expansion $S_T(\beta)$ of S around $\beta^{(n)}$ is shown in Equation 2.8

$$S_T(\beta) = S_T(\beta^{(n)} + \delta) \approx S(\beta^{(n)}) + S'(\beta^{(n)})\delta + \frac{1}{2}S''(\beta^{(n)})\delta^2 \tag{2.8}$$

In the one-dimensional problem, Newton's method attempts to construct a sequence $\beta^{(n)}$ from an initial guess $\beta^{(0)}$ that converges towards some value β^* satisfying $S'(\beta^*) = 0$. This β^* is a stationary point of function S. For finding δ such that $\beta^{(n)} + \delta$ is a stationary point. We seek to solve the equation that sets the derivative of this last expression with respect to δ equal to zero:

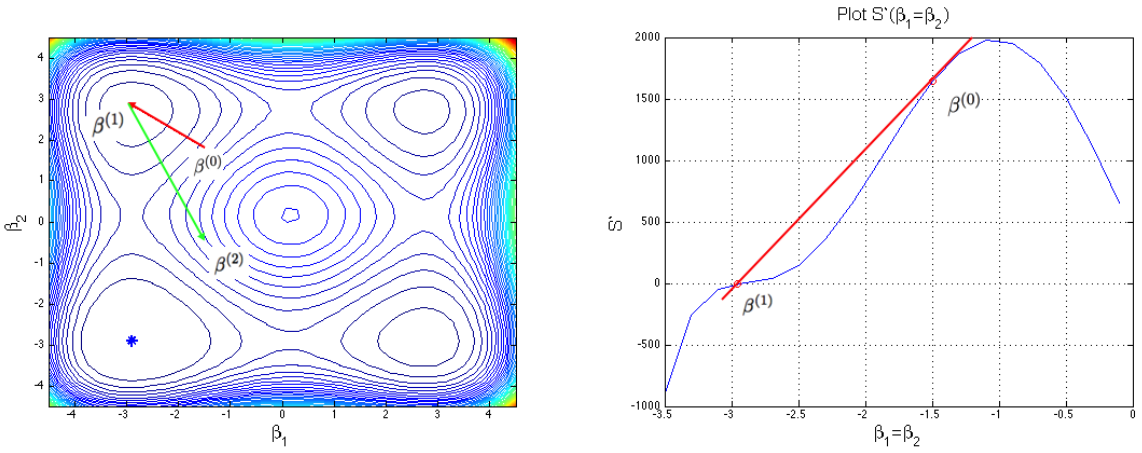
$$0 = \frac{d}{d\delta}\left(S(\beta^{(n)}) + S'(\beta^{(n)})\delta + \frac{1}{2}S''(\beta^{(n)})\delta^2\right) = S'(\beta^{(n)}) + S''(\beta^{(n)})\delta \tag{2.9}$$

Therefore, the Equation 2.9 can be simplified as follows.

$$\beta^{(n+1)} = \beta^{(n)} + \delta = \beta^{(n)} - \frac{S'(\beta^{(n)})}{S''(\beta^{(n)})} \tag{2.10}$$

If the first derivative S' is flat (second derivative $S'' \approx 0$) at some iterations, Newton's method may not work due to the optimizer shooting out of the solution area. For avoiding the failure of Newton's method, only the improved result of S ($S^{(n+1)} < S^{(n)}$)will be accepted during the optimization iteration.

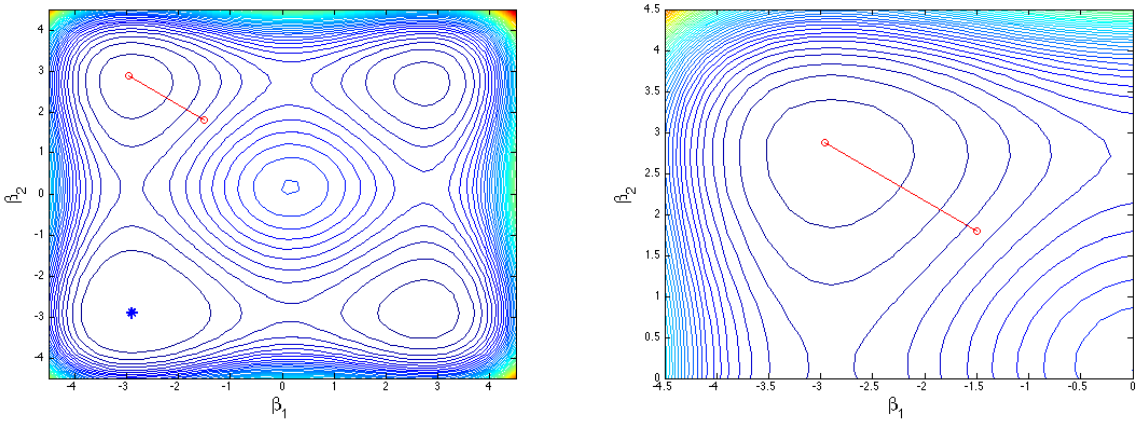
Figure 2.9 shows one iteration after applying the Newton's method (using second derivative information for providing the search direction) for optimizing modified Styblinski-Tang function. In the Figure 2.9a, $\beta^{(0)}...\beta^{(2)}$ are the result parameters of each search step. Red arrow is the search direction for first step. Because the first derivative curve is very flat ($S'' \approx 0$) at $\vec{\beta}^{(1)}$ point, the next iteration shot very far form the minimum value by Newton's method. Based the updating rule, this step is rejected. The section along the optimizer moving direction is shown in Figure 2.9b. The blue line is the section curve of the first derivative of MST (S'), and the red lines are the iteration steps. As shown in the figure, there is only one Newton's iteration applied to the objective function, because the S' curve is very flat after one iteration. This means the $S'' \approx 0$ and the next Newton's iteration will shoot out of the solution area (increase the objective a lot). Therefore, the next iterations are declined.



(a) Contour of the modified Styblinski-Tang (b) Section curve of the first derivative of MST

Figure 2.9: Search directions provided by Newton's method (second derivative information)

The trajectory (red lines) of whole Newton's method optimization process is shown in Figure 2.10. For the Newton's method, the optimization process stopped at local minimum on the up left corner as same as gradient decent, because the steps only move to the decreasing objective directions. Although the number of iterations is reduced further comparing with gradient decent method, there is no iterations around the minimum value for improving the accuracy of the result. There is only 1 iterations till converge. Because the first and second derivatives information for $\vec{\beta}$ (2 design parameters) need to be calculated in each iteration, there are total $(1+2+4) = 7$ function evaluations in each iteration. The total function evaluations for gradient decent method is 7.



(a) Trajectory of convergence (b) Zoom in the figure

Figure 2.10: Trajectory of objective function value based on Newton's method

The log of objective function value after each iteration is shown in Figure 2.11.

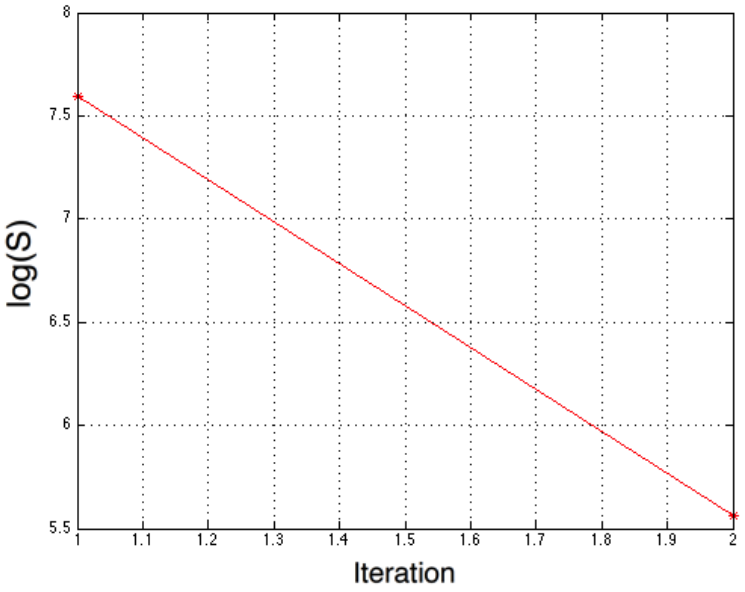


Figure 2.11: Log of objective function value at each iteration based on Newton's method

Newton's method optimized the step size for each iteration, but it not robustness enough around the first derivative flat area. For improving the robustness around $S''approx0$

area, the Levenberg-Marquardt algorithm will be introduced in the next section.

2.3.3 Levenberg-Marquardt Method

The Levenberg-Marquardt method[113] is an optimization method which is used to minimize least square problems. It is a combination of Newton's method and gradient descent, making it both fast and robust.

For the multiple-dimensional ($\vec{\beta}$) problem, one starts from Newton's method as an iterative method for finding the root of a differentiable function S. In optimization, Newton's method is applied to the derivative S' of a twice differentiable function S to find the root of S'. Once the root of S' is found, the maximum or minimum value of the objective function S is found.

Newton's method, to drive S' to zero, can be written as Equation 2.11

$$\vec{\beta}^{(s+1)} - \vec{\beta}^{(s)} = -[H(\vec{\beta}^{(s)})]^{-1} \cdot \vec{g}(\vec{\beta}^{(s)}) \tag{2.11}$$

where $\vec{\beta}$ are the design variables, $\vec{g}$ is the first derivative of S in vector format (here, $\vec{g}$ only contains one item due to S just have one square item), and H is the Hessian matrix of S. For the current problem, $\vec{\beta} = (\beta_1, \beta_2)$. For simplicity, it is useful to define as Equation 2.12

$$\vec{\delta} \equiv \vec{\beta}^{(s+1)} - \vec{\beta}^{(s)} \tag{2.12}$$

We defined the number of square terms in objective function is t ($t = 1$ for the modified Styblinski-Tang function) and each square term is q_l, $l = 1...t$. By taking the advantage of the least squares structure of objective function, gradient and Hession matrix can be

written as Equation 2.13 and 2.14

$$g_j = 2\sum_{l=1}^{t} q_l \frac{\partial q_l}{\partial \beta_j} \tag{2.13}$$

$$H_{jk} = 2\sum_{l=1}^{t} \left(\frac{\partial q_l}{\partial \beta_j}\frac{\partial q_l}{\partial \beta_k} + q_l \frac{\partial^2 q_l}{\partial \beta_j \partial \beta_k}\right) \tag{2.14}$$

Now define $J \equiv \partial q_l / \partial \beta_j$, which is the Jacobian matrix of $\vec{q}$. (Recall the $\vec{g}$ is the gradient of q^2.) Marquardt assumed that the second-order derivative terms of H could be ignored. This assumption is based on Equation 2.15.

$$\left| q_l \frac{\partial^2 q_l}{\partial \beta_j \partial \beta_k} \right| \ll \left| \frac{\partial q_l}{\partial \beta_j}\frac{\partial q_l}{\partial \beta_k} \right| \tag{2.15}$$

Ignoring the second-order derivative terms may be valid in two cases, for which convergence is to be expected. [114]

- The function values q_l are small in magnitude, at least around the minimum.
- The functions are only "mildly" non linear, so that $\frac{\partial^2 q_l}{\partial \beta_j \partial \beta_k}$ is relatively small in magnitude.

After ignoring the second-order derivative terms of H, the Hession matrix can be expressed as Equation 2.16

$$H_{jk} = 2\sum_{l=1}^{t} J_{lj} J_{lk} \tag{2.16}$$

Combining these gives a result that the iteration function can be written as Equation 2.17

$$\vec{\delta} = -(J^T J)^{-1} \cdot J^T \vec{q} \tag{2.17}$$

If the first derivative S' is flat (second derivative $S'' \approx 0$), Newton's method may not work; hence Marquardt suggested adding a term that employs ideas from the gradient descent method (since it only uses first derivatives). This is done by adding a damping parameter, λ, to the iteration function that can be changed based on the result in each step. The function is shown as Equation 2.18

$$\vec{\delta} = -(J^T J + \lambda I)^{-1} \cdot J^T \vec{q} \tag{2.18}$$

where I is the identity matrix.

When $\lambda \to 0$, the added term vanishes and the technique reverts to Newton's method. When λ becomes large, the scheme becomes the gradient descent method. This improves the robustness of the algorithm when the initial values are far from the final minimum.

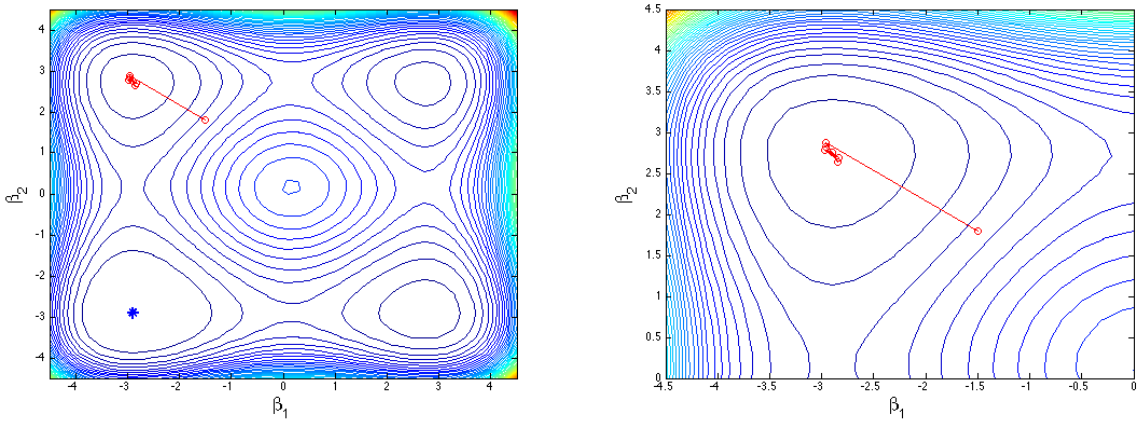
Marquardt suggested starting with a small value for λ. If the results of the previous step improves the objective function, (that is, $||\vec{e}^{(new)}|| < ||\vec{e}^{(old)}||$) $\vec{\beta}$ is incremented by $\vec{\delta}$ and the the value of λ is decreased (say by a factor of 2) and the method continues. If the method (unfortunately) increases the objective function, the step is discarded and λ is increased.

Marquardt also provided the insight that one can scale each component of the gradient according to the curvature so that there is larger movement along the directions where the gradient is smaller. This avoids slow convergence in the direction of small gradient. Therefore, the identity matrix I was replaced with the diagonal matrix consisting of the diagonal elements of $J^T J$, yielding, and now the iteration function as Equation 3.12

$$\vec{\delta} = -(J^T J + \lambda \cdot \mathrm{diag}(J^T J))^{-1} \cdot J^T \vec{q} \tag{2.19}$$

There are two stopping rules for interrupting the iteration. One is the number of iterations exceeds the maximum number of iterations, which is defined by the user. The other one is when the $||\vec{\delta}||$ becomes smaller than a specified tolerance, ϵ.

The trajectory (red lines) of optimization process is shown in Figure 2.12. Comparing with the gradient decent and Newton's algorithms, Levenberg-Marquardt algorithm overcomes the the two algorithm's shortness and combines them together. However, as shown in the figure, the iteration is also can not reach the global minimum value due to the gradient-based searching direction and monotonic updating rule. Due to LM takes the structural advantages of least square problem and assumes ignored the second derivative terms in Hession matrix, the number of function evaluations for each iteration is (1+2) = 3. There are total 20 iterations in LM algorithm. So the total number of function evaluations is 60.



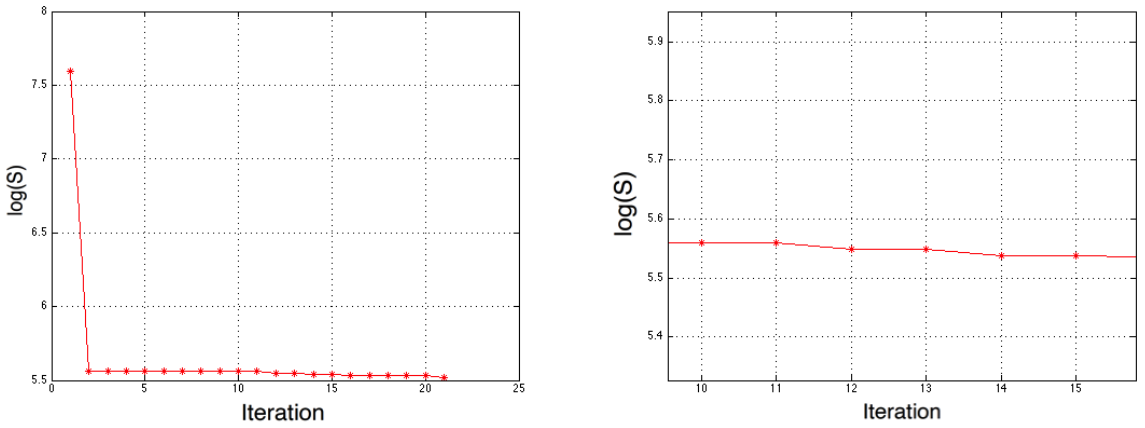
(a) Trajectory of convergence

(b) Zoom in the figure

Figure 2.12: Trajectory of objective function value based on LM method

The log of objective function value after each iteration is shown in Figure 2.13. The λ values during the optimization process is shown is Figure 2.14. If the objective value reduced after one iteration, the λ is reduced also for making it more Newton's like (more

efficiency). If the objective value did not reduce after one iteration, the λ is increased for making it more gradient decent like (more robustness).



(a) Objective function value (b) Zoom in the figure

Figure 2.13: Log of objective function value at each iteration based on LM

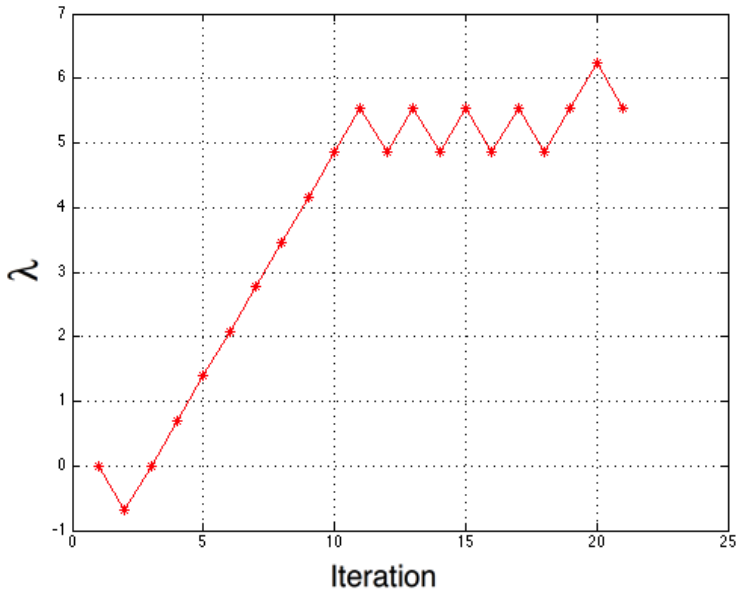


Figure 2.14: λ values at each iteration based on LM

LM method takes the both advantages of gradient decent and Newton's methods, it is still converges at local minimum due to the monotonic updating rule. Therefore, the idea

borrowed from simulated annealing is combine into LM method for overcoming the local minimum problem.

2.3.4 Improved Levenberg-Marquardt Method

In LM method, sometimes the optimization processes is slow, because the parameters have not been updated during many iterations; this means that many steps were rejected and λ was increased. The original LM method also has a high probability of getting stuck in a local minimum due to the monotonic updating rule.

To circumvent these problems, a modification to the LM method is developed, which borrows some ideas from the simulated annealing method (SA). Here, recall the definition of simulated annealing (SA) in Chapter 2.1.3. SA is a heuristic optimization method that overcomes the local minimum problem somehow. It interprets slow cooling as a slow decrease in the probability of accepting worse solutions as it explores the solution space. Accepting worse solutions is a fundamental property of heuristics because it allows for a more extensive search for the optimal solution.

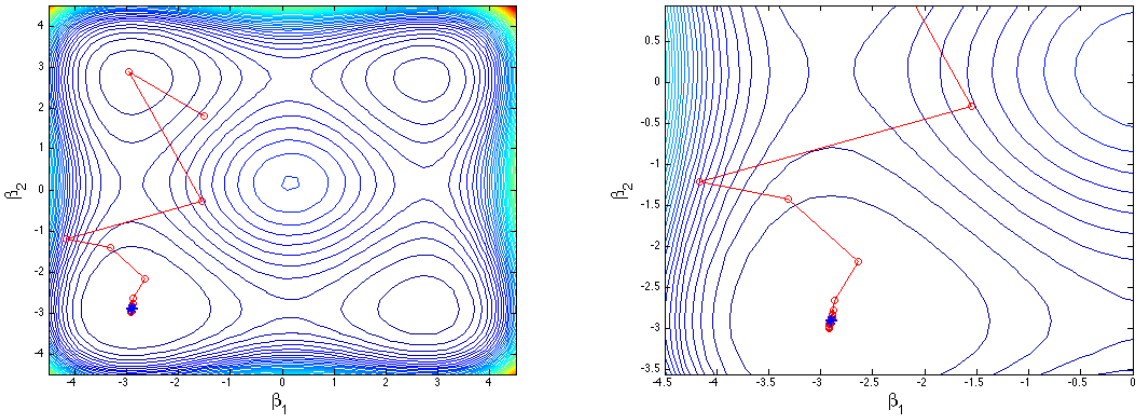
After borrowing the idea from SA, candidate solutions are randomly generated and accepted if the new objective is "not too much worse" than the previous objective. The definition of "not too much worse" starts rather loosely and then tightens up as the method continues This idea is inspired by the annealing process for metals and results in the SA method being less prone to getting stuck in a local minimum.

Here the same idea is added to the original LM method. Specifically, instead of accepting steps only if the objective improves, one can accept new results that are "not too much worse", or a step is accepted if

$$||\vec{e}^{(new)}|| < e^{||\vec{\delta}||} \, ||\vec{e}^{(old)}|| \tag{2.20}$$

Since $||\vec{\delta}||$ approaches 0 as the solution is neared, the factor $e^{||\vec{\delta}||}$ approaches 1, and only better steps are accepted during the endgame (as was done in the SA method). The new algorithm is named as Improved Levenberg-Marquardt (ILM) algorithm.

The trajectory (red lines) of optimization process is shown in Figure 2.15. Because the Improved Levenberg-Marquardt borrows the idea form simulated annealing that "accept the iteration result which is not too bad", the optimization process has the property that goes out from local minimum value and moves to the global minimum value. As shown in figure, the global minimum value is gotten after using ILM algorithm and the parameter $\vec{\beta}$ is much equal to the correct values after 100 iterations. The number of function evaluations for ILM methods is $(1 + 2) \times 100 = 300$.

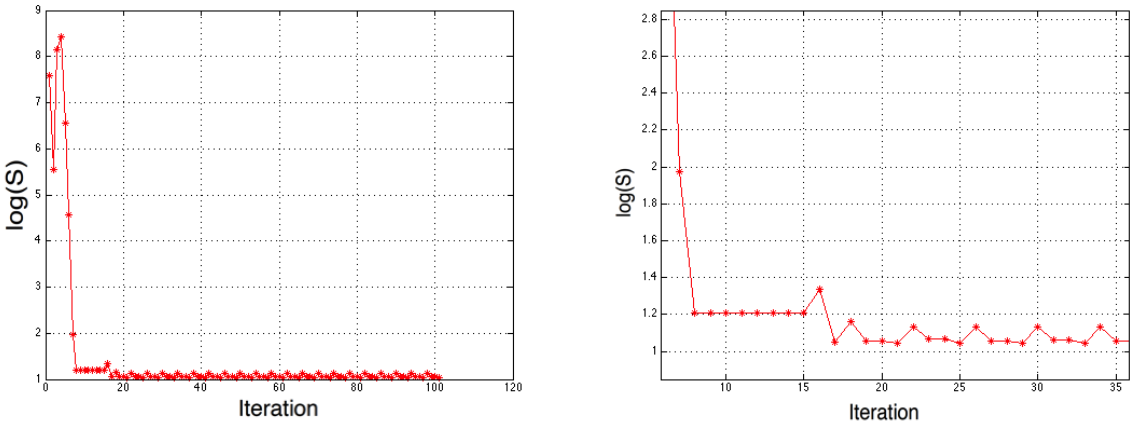


(a) Trajectory of convergence

(b) Zoom in the figure

Figure 2.15: Trajectory of objective function value based on ILM method

The log of objective function value after each iteration is shown in Figure 2.16. The λ value at each iteration is shown in Figure 2.17



(a) Objective function value (b) Zoom in the figure

Figure 2.16: Objective function value at each iteration based on ILM

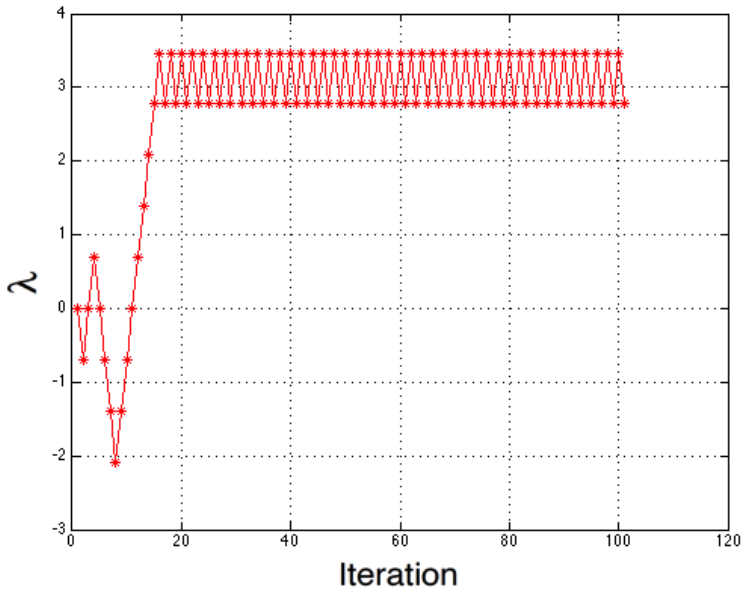
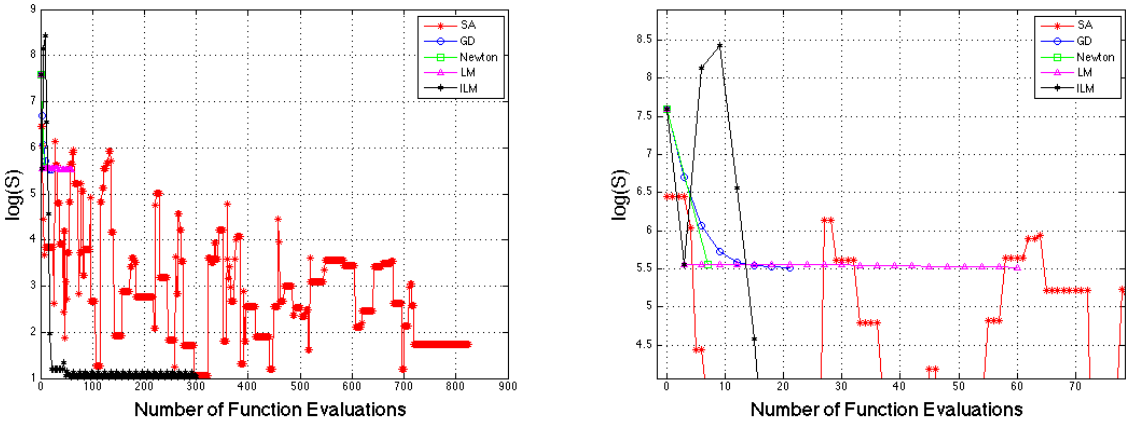


Figure 2.17: λ values at each iteration based on ILM

For comparing the converge speed for different optimization algorithms base on the number of function evaluations, the objective function value histories are shown in Figure 2.18. As shown in the figure, the SA method can reached at the global minimum area, but there are too many function evaluations here. The gradient decent, Newton's and

LM method are converged faster than SA method, but they are get stuck at the local minimum value. For taking the advantages of these 4 algorithms, the ILM method can overcome the local minimum problem and the speed of convergence is fast.



(a) Objective function value

(b) Zoom in the figure

Figure 2.18: Objective function value for applying different optimization algorithms

The ILM is provided a method that optimizer has a chance to get out of the local minimum, but there is no guarantee that the global minimum can be reached at the end. Comparing with the other three algorithms, if the initial starts from the local minimum areas, gradient decent, Newton's and original LM are all moving to the local minimum value (due to the iterations are monotonically decreasing). However, for the ILM method, there is a chance that the optimizer moves out of the local minimum area (due to the "uphill" steps are taken at the beginning) and gets into the global minimum area (not guarantee because optimizer maybe moves to another local minimum). For testing the performance of the four algorithms, 20 random initial guess problems are run and the results are shown in below:

Table 2.1: Comparison of Performance for Different Optimization Algorithms by 20 Random Initial Guess

Initial	SA	Gradient decent	Newton's method	LM	ILM
(-2.80, 2.98)	(-2.98, 2.77)	(-2.80, 2.98)	(-2.80, 2.98)	(-2.89, 2.74)	(2.72, -2.91)
(0.02, 1.06)	(-2.73, 2.38)	(0.02, 1.06)	(0.02, 1.06)	(2.90, 2.74)	(-2.87, -2.90)
(1.99, 0.74)	(2.73, -3.11)	(1.99, 0.74)	(1.99, 0.74)	(2.76, 2.76)	(-2.90, -2.85)
(3.51, 3.75)	(2.66, 2.76)	(2.93, 2.90)	(2.91, 2.87)	(2.72, 2.72)	(2.74, -3.30)
(4.13, -1.92)	(2.87, -2.72)	(2.55, -2.74)	(2.44, -2.85)	(2.74, -2.89)	(4.78, 2.94)
(0.42, 2.31)	(-3.04, 2.82)	(0.42, 2.31)	(0.42, 2.31)	(2.76, 2.76)	(2.75, -2.92)
(-3.25, 2.28)	(-3.16, 2.78)	(-2.52, 2.80)	(-3.25, 2.28)	(-2.88, 2.74)	(2.72, 2.72)
(-3.15, -1.07)	(-2.92, -2.73)	(-2.89, -2.88)	(-2.98, -2.98)	(-2.88, -2.88)	(-3.30, 2.74)
(-2.18, 0.61)	(-2.68, 2.82)	(-2.18, 0.61)	(-2.18, 0.61)	(-2.88, 2.74)	(-2.87, -2.87)
(3.06, -3.81)	(2.69, -2.58)	(2.90, -3.11)	(2.89, -3.08)	(2.75, -2.88)	(2.74, -3.30)
(-2.21, -4.01)	(-2.80, -2.77)	(-2.88, -2.90)	(-2.85, -2.88)	(-2.87, -2.89)	(2.82, -3.03)
(2.82, 0.27)	(2.63, 2.65)	(2.82, 0.27)	(2.82, 0.27)	(2.92, 2.77)	(-2.86, -2.88)
(-2.30, 2.51)	(-2.83, 3.01)	(-3.20, 2.87)	(-2.30, 2.51)	(-2.88, 2.74)	(-3.30, 2.74)
(3.86, 3.90)	(2.71, 2.65)	(3.15, 3.15)	(3.14, 3.14)	(2.76, 2.76)	(2.69, 2.69)
(-1.35, -3.33)	(-2.96, -2.67)	(-2.88, -2.90)	(-2.96, -2.92)	(-2.88, -2.89)	(-2.94, -2.91)
(-2.73, 0.61)	(-3.05, 2.51)	(-2.73, 0.61)	(-2.73, 0.61)	(-2.88, 2.74)	(-2.90, -2.85)
(-2.24, -0.27)	(-2.82, -2.39)	(-2.24, -0.27)	(-2.24, -0.27)	(-2.90, -2.87)	(-2.91, -2.91)
(1.04, -4.39)	(2.80, -3.05)	(2.60, -3.12)	(2.78, -3.20)	(2.74, -2.88)	(2.82, -3.04)
(-0.24, -1.46)	(-2.92, -3.02)	(-2.87, -2.90)	(-0.24, -1.46)	(-2.92, -2.90)	(-2.85, -2.89)
(-1.33, -3.04)	(-3.05, -2.95)	(-2.89, -2.90)	(-2.86, -2.89)	(-2.88, -2.89)	(-2.85, -2.89)

For simplifying the Table 2.1, each point is expressed as the color representing the different domains. The domain $(-,+)$ is represented by B (blue). The domain $(+,+)$ is represented by R (red). The domain $(-,-)$ is represented by G (green). The domain $(+,-)$ is

represented by P (pink). The global minimum value is contained in domain $(-,-)$ as green. The color domains are shown in Figure 2.19

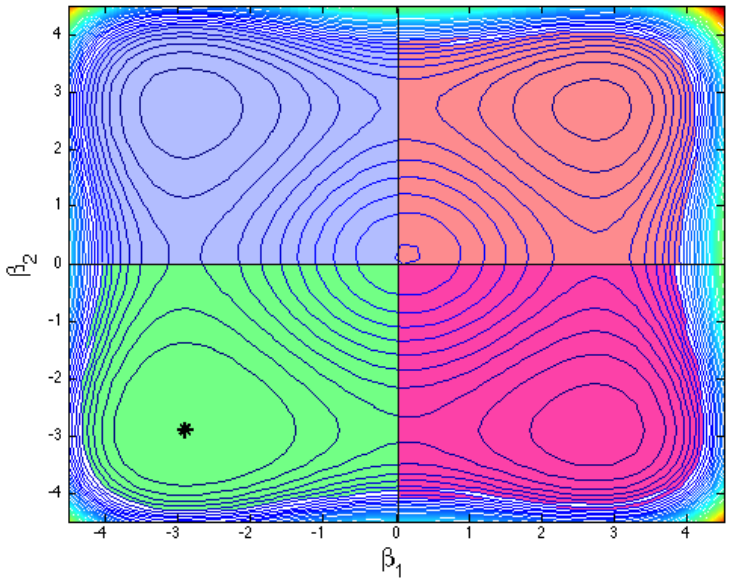


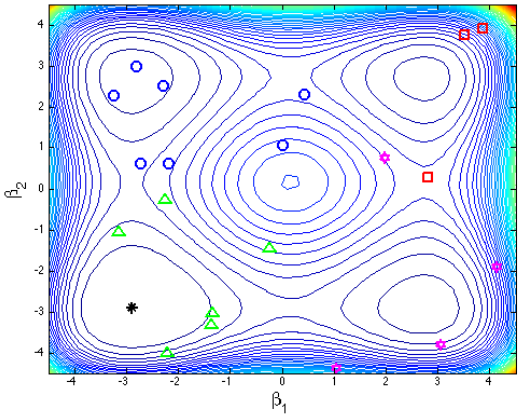
Figure 2.19: Domains represented by different colors

Table 2.2: Comparison of Performance for Different Optimization Algorithms (express the results by color, B(blue), G(green), P(pink), R(red))

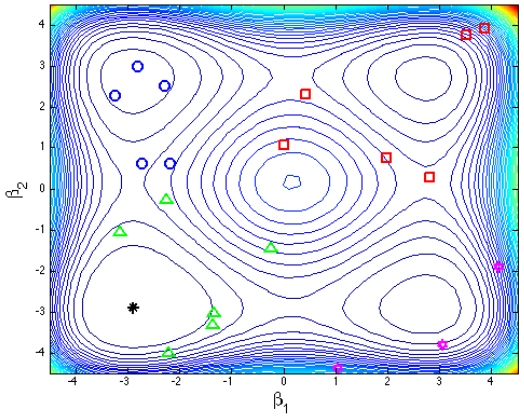
Initial	SA	Gradient decent	Newton's method	LM	ILM
B	B	B	B	B	P
R	B	R	R	R	G
R	P	R	R	R	G
R	R	R	R	R	P
P	P	P	P	P	R
R	B	R	R	R	P
B	B	B	B	B	R
G	G	G	G	G	B
B	B	B	B	B	G
P	P	P	P	P	P
G	G	G	G	G	P
R	R	R	R	R	G
B	B	B	B	B	B
R	R	R	R	R	R
G	G	G	G	G	G
B	B	B	B	B	G
G	G	G	G	G	G
P	P	P	P	P	P
G	G	G	G	G	G
G	G	G	G	G	G

The Table 2.2 also can be expressed as Figure 2.20. There are only comparing the differences between SA, LM and ILM, because the gradient decent, Newton's and LM are

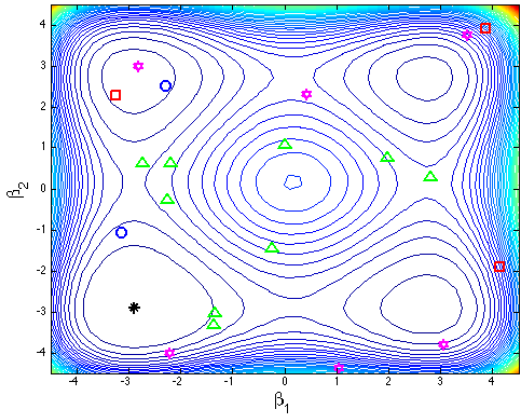
the same monotonic updating rule which only "downhill" steps are accepted.



(a) Optimization results based on SA method



(b) Optimization results based on LM method



(c) Optimization results based on ILM method

Figure 2.20: Performance of SA, LM and ILM optimization algorithms

From the 20 tests starting from random initial guess, the SA method can get out of the local minimum area, but there is not much possibility to converge to global minimum value. There are 6 green points (stop at global minimum area) in Figure 2.20a. For LM method, all points are stay in the initial local minimum area and converge at local minimum value. There are 6 green points in Figure 2.20b. For the ILM method, some points can get out of the local minimum value are, and converge at global minimum value. There area 9 green points in Figure 2.20c. So, ILM has 50% more percent possibility to converge at

global minimum. Therefore, the ILM algorithm will be used as the optimization method for generating the parametric geometry model from a cloud points.

2.4 Application of ILM Algorithm

Now that the optimization techniques have been discussed for finding the minimum value in mathematical problem, one can be also applied in other fields, such as: engineering design [88–93], economics [94–97], energy saving [115, 115–119] and so on. In this thesis, the new optimization technique can overcome the local minimum problem during optimizing the nonlinear objective function, and also have the advantage of least square problem's structure for improving the efficiency of iterations. In this section, this new algorithm will be applied to traffic assignment problem for testing the accuracy, efficiency and robustness.

Here, the Traffic Equilibrium Problem (TEP) is used as demonstration problem. Traffic assignment is a core component in transportation planning and real-time applications in optimal routing, signal control, and traffic prediction in traffic networks. It models the flow pattern in a network given a set of travel demands between the origin-destination (OD) pairs. The most widely used route choice model is the user-equilibrium (UE) principle. This UE assignment finds the flow pattern by allocating the OD demands to the network in such a way that no drivers can unilaterally change routes to achieve better travel times. [104]

2.4.1 Problem Statement

To test the ILM algorithm, we apply it to a simple network with five nodes, five links, and two OD pairs, as given in Figure 2.21.The travel demand between OD pair (1, 5) is

15 units and (2, 5) is 18.75 units. There are four possible routes: route 1 (on links 1 → 4 → 5), route 2 (on links 1 →3 → 5), route 3 (on links 2 → 4 → 5) and route 4 (on links 2 → 3 → 5). The link performance function is the standard BPR function:

$$t_a = \alpha_a(1 + 0.15(\frac{\nu_a}{C_a})^4) \tag{2.21}$$

where, t_a, α_a, ν_a and C_a are the travel time, free-flow travel time, flow, and capacity of link a, respectively. Link characteristics are provided in Table 2.3.

Table 2.3: Link Characteristics of the Example Network

Link	Free-flow travel time (α_a)	Capacity (C_a)
1	15	10
2	10	20
3	10	20
4	15	15
5	10	30

Specially, we assume that a 5-unit cost was added to route 4 (link sequence: 2-3-5) while no cost was added to the other routes, namely, $\lambda_4^{25} = 5$, $\lambda_1^{15} = \lambda_2^{15} = \lambda_3^{25} = 0$

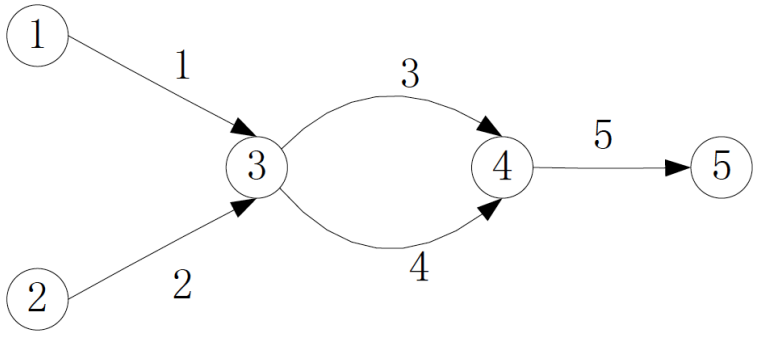


Figure 2.21: A simple network for traffic equilibrium problem

2.4.2 Objective Function and Algorithm Application

The objective function proposed by Hong [104] is defined as:

$$S(\vec{x}) = \sum_{rs}\sum_{p}\frac{1}{2}\left\{\sqrt{(f_p^{rs})^2 + (\eta_p^{rs} - \pi^{rs})^2} - (f_p^{rs} + \eta_p^{rs} - \pi^{rs})\right\}^2 + \sum_{rs}\frac{1}{2}\left\{\sqrt{(\pi^{rs})^2 + (\sum_{p} f_p^{rs} - q^{rs})^2} - ((\pi^{rs})^2 + (\sum_{p} f_p^{rs} - q^{rs})^2)\right\}^2 \quad (2.22)$$

where, η_p^{rs} is the route cost function as:

$$\eta_p^{rs}(\vec{f}) = \lambda_p^{rs} + \sum_{a}\delta_{p,a}^{rs} t_a \quad (2.23)$$

π^{rs} is the minimal route cost: $\pi^{rs} = min_p \eta_p^{rs}, \forall rs \in RS$. where, RS is the set of OD pairs for the whole network. rs is an OD pair, $rs \in RS$. f_p^{rs} is the flow on route p between OD pair rs. η_p^{rs} is the route cost (or dis-utility) on route p between OD pair rs. q^{rs} is the demand between OD pair rs.

2.4.3 Results

After applying the ILM algorithm for optimizing the TEP, the results comparing with the original value are shown in Table 2.4 and Table 2.5

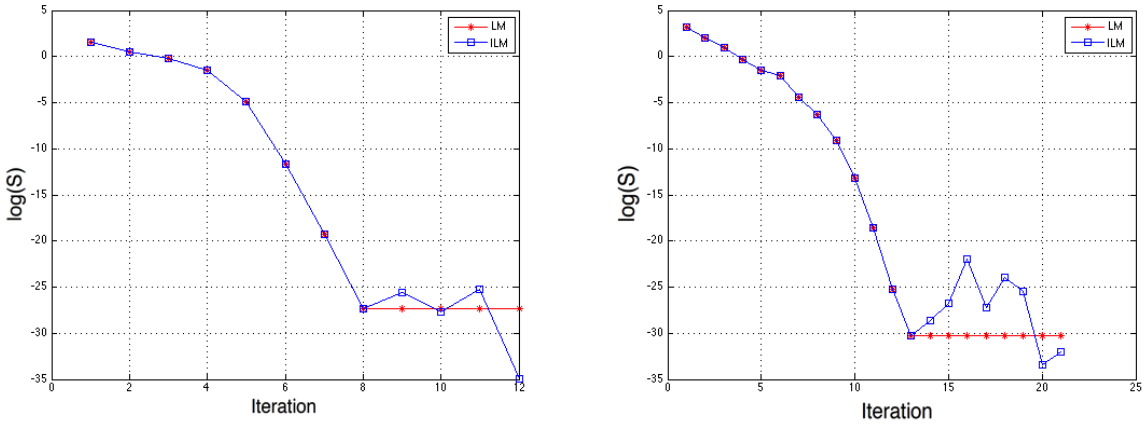
Table 2.4: Comparison of Link Flows for the Additive and Route-specific Cost Models

	Original values		ILM results	
Link	Additive route cost	Route-specific cost	Additive route cost	Route-specific cost
1	15	15	14.96	14.96
2	18.75	18.75	18.75	18.75
3	27.14	20.12	27.14	20.11
4	6.61	13.63	6.61	13.63
5	33.75	33.75	33.75	33.75

Table 2.5: Comparison of Route Flows for the Additive and Route-specific Cost Models

		Original values				ILM results			
		Additive route cost		Route-specific cost		Additive route cost		Route-specific cost	
OD	Route	Route flow	Route cost	Route flow	Route cost	Route flow	Route cost	Route flow	Route cost
1,5	1	2.52	53.87	0.00	50.33	2.52	53.75	0.00	50.20
	2	12.48	53.87	15.00	50.33	12.48	53.75	15.00	50.20
2,5	3	4.09	38.65	13.63	40.10	4.09	38.65	13.59	40.09
	4	14.66	38.65	5.12	40.10	14.66	38.65	5.16	40.09

As shown in the tables, the results from ILM algorithm are accurate comparing with the correct values. And from the convergence Figure 2.22, comparing with original LM algorithm, ILM reached smaller value of objective function. This means ILM can overcome the local minimum value problem.



(a) Convergence characteristics of additive cost models

(b) Convergence characteristics of route-specific cost models

Figure 2.22: Convergence characteristics comparison between LM and ILM algorithms

Chapter 3

Fitting a Simplified Model to a Cloud of Points

As introduced in the first Chapter, the objective of this work is to find the design parameters, which are associated with a parametric solid model, that best-fits a cloud of points. In Chapter 2, several optimization schemes were discussed, culminating in the discussion of the Improved Levenberg-Marquardt (ILM) scheme for solving least-squares problems. In this Chapter, the ILM scheme is applied to the parametric model problem. It is assumed in this Chapter that one starts with a reasonable initial guess for all independent variables. Techniques for finding an initial guess are described fully in Chapter 4.

In section 3.1, the optimization problem is defined. Then in section 3.2, each of the optimization techniques described in Chapter 2 (gradient descent, Newton's method, Levenberg-Marquardt, and Improved Levenberg-Marquardt) are applied to a simplifies fitting problem, namely configuration that is composed of a single super-ellipse. Section 3.3 discusses the sparse matrix procedures that are used in order to minimize the computer resources needed within the optimization scheme.

3.1 Demonstration Problem

The basic optimization techniques area demonstrated by applying them to a configuration that contains a single super-ellipse. A super-ellipse is a generalization of an ellipse, where the power, r, can be something other than 2. In particular, the equation for a super-ellipse is given by

$$\left|\frac{x}{a}\right|^r + \left|\frac{y}{b}\right|^r = 1 \tag{3.1}$$

The reason for choosing a super-ellipse as the demonstrated geometry is that it closely approximates the cross-sections of many aircraft fuselages, as shown in Figure 3.1.

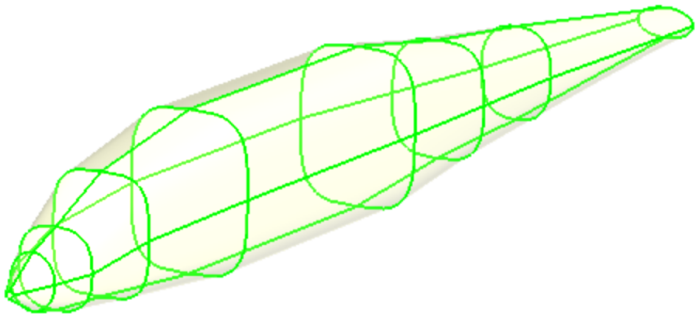
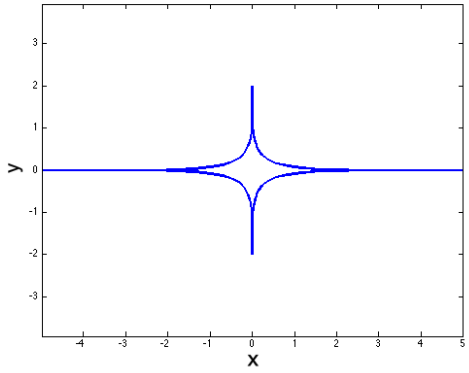
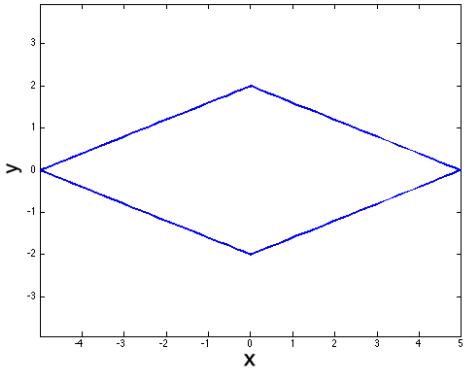


Figure 3.1: Cross sections of fuselage

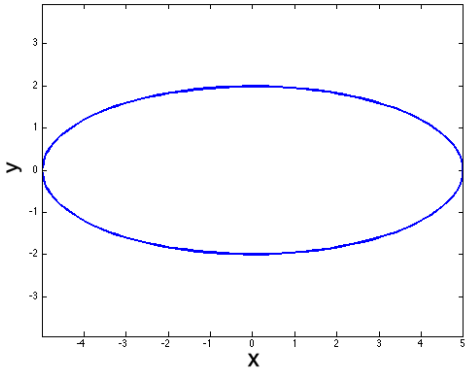
To see the generality of the super-ellipse, Figure 3.2 shows the results of choosing several values for r, from 0.3 up to 5. Notice that small values of the power r yield cusped configurations, whereas large values of the power r yield shapes that resemble a rectangle with rounded corners.



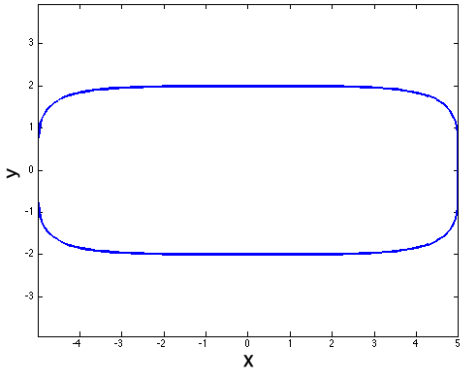
(a) Super-ellipse when the power $r = 0.3$



(b) Super-ellipse when the power $r = 1$



(c) Super-ellipse when the power $r = 2$



(d) Super-ellipse when the power $r = 5$

Figure 3.2: Super-ellipses based on the different powers r

3.1.1 Design Parameters

Design parameters are the values that can be used to generate a specific super-ellipse. Here, the elements of the design parameter vector, $\vec{d}$, are the major and minor radii, a and b, the super-ellipse power, r, the center point, $(\delta x, \delta y)$, and the rotation angle θ. In the discussion that follows, the length of $\vec{d}$ is n; therefore $n = 6$ when $\vec{d} = (a, b, r, \delta x, \delta, \theta)$.

The defining parametric function for a super-ellipse centered at the origin is given by

$$\begin{aligned} x_t &= |\cos u|^{2/r} \cdot a \cdot \text{sign}(\cos u) \\ y_t &= |\sin u|^{2/r} \cdot b \cdot \text{sign}(\sin u) \end{aligned} \tag{3.2}$$

where u is a parametric coordinate around the super-ellipse, with $0 \leq u \leq 2\pi$.

Homogeneous Coordinates

Homogeneous coordinates were introduced by August Ferdinand Mobius in 1827.[120] It is a system of coordinates used in projective geometry, as Cartesian coordinates are used in Euclidean geometry. While homogeneous coordinates are frequently used in computer graphics because of their ability to convert 3D coordinates onto a plane (with perspective), they are used here because they allow one to specify translations, rotations, and scalings via simple matrix multiplications. This makes formulas involving homogeneous coordinates simpler than their Cartesian counterparts.

As an example, consider the translation matrix is shown in Equation 3.3,

$$\begin{bmatrix} f_x \\ f_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} \tag{3.3}$$

where $(\delta x, \delta y)$ are the translation distances. The result of such a translation is shown in Figure 3.3.

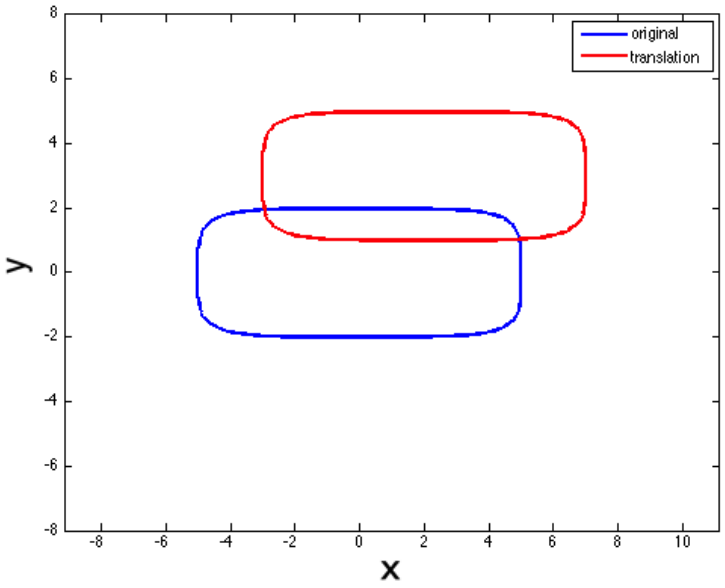


Figure 3.3: Super-ellipse after translation, with $\delta x = 2$ and $\delta y = 3$

A rotation matrix is given by 3.4

$$\begin{bmatrix} f_x \\ f_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} \tag{3.4}$$

where θ is the counter-clockwise rotation about the z axis. Figure 3.4 shows the results of such a rotation.

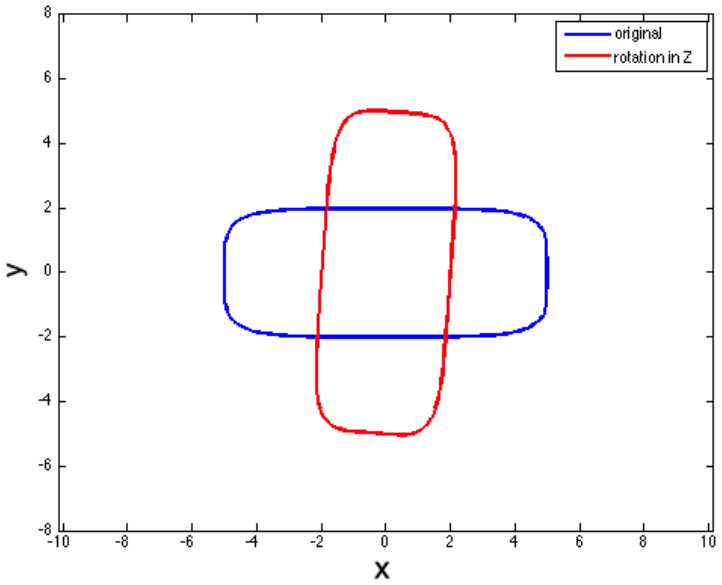


Figure 3.4: Super-ellipse after rotation in z axis, with $\theta = 1.5$ (radians).

In a similar way, a scaling matrix can be defined by

$$\begin{bmatrix} f_x \\ f_y \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} \tag{3.5}$$

where S_x and S_y are the scaling in the x and y directions, respectively. Figure 3.5 shows such a scaling. Note that scaling is not used in the current application since one can directly specify the semi-major- and semi-minor-axis lengths (a and b) in the super-ellipse formula.

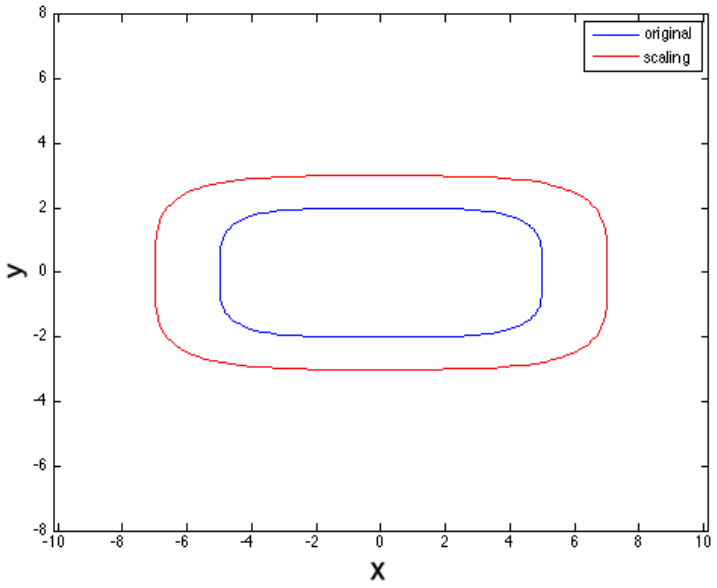


Figure 3.5: Super-ellipse after scaling, with $S_x = S_y = 1.2$

To apply these transformation, one starts with the coordinates on the super-ellipse centered at the origin, as given in Equation 3.2. These coordinates can then be transformed (translated and rotated) using homogeneous coordinates, as in Equation 3.6. An example of a "general" super-ellipse ins shown in Figure 3.6.

$$\begin{bmatrix} f_x \\ f_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} \tag{3.6}$$

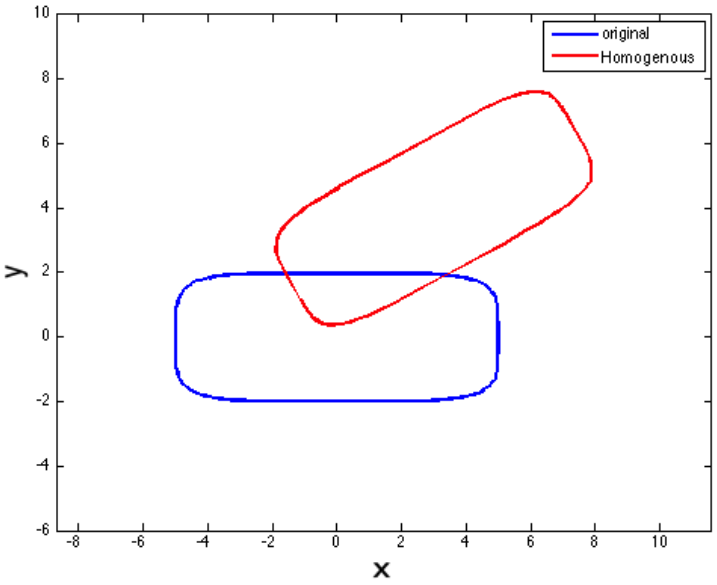
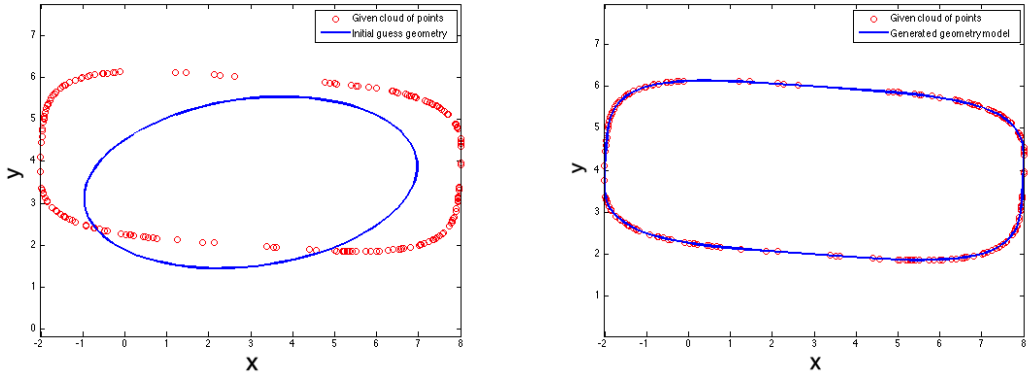


Figure 3.6: Super-ellipse after applying homogeneous coordinates

3.1.2 Least Square Objective Function

For the current demonstration problem, consider the fitting of a super-ellipse to a cloud of points, as shown in Figure 3.7, where the blue line in Figure 3.7a is the super-ellipse generated from the initially guessed design parameters $\vec{d}$, and the red points are the points in the cloud.

The objective here is to vary the design parameters so as to find the super-ellipse shown in Figure 3.7b, which clearly is a better fit to the cloud of points than the original super-ellipse.



(a) Cloud of points (red) and initial guess (blue)

(b) Cloud of points (red) and final model (blue)

Figure 3.7: Sample problem, demonstrated with a super-ellipse.

The objective function, that is the quantity to be minimized, is the least-square distances between the points in the cloud and the surface of the parametric model (in the current case, the surface of the super-ellipse). For each point i in the cloud, one can define the fitting error as

$$e_{(i)} = (x_{p(i)} - f_x(\vec{d}, u_i))^2 + (y_{p(i)} - f_y(\vec{d}, u_i))^2 \tag{3.7}$$

Here, $x_{p(i)}$ and $y_{p(i)}$ are the x- and y-coordinates of the ith point in the cloud, $f_x(\vec{d}, u_i)$ and $f_y(\vec{d}, u_i)$ are the coordinates of a point on the super-ellipse associated with a parametric coordinate u_i.

Since there are m points in the cloud, the objective function can be written as the sum of the distances $e_{(i)}$, as in

$$S = \sum_{i=1}^{m} e_{(i)} = \sum_{i=1}^{m} \left\{ (x_{p(i)} - f_x(\vec{d}, u_i))^2 + (y_{p(i)} - f_y(\vec{d}, u_i))^2 \right\} \tag{3.8}$$

To simplify the above as a single sum, one can define

$$q_l = \begin{cases} x_{p(l)} - f_x(\vec{d}, u_l) & \text{for} \quad l <= m \\ y_{p(l-m)} - f_y(\vec{d}, u_{l-m}) & \text{for} \quad l > m \end{cases} \tag{3.9}$$

Then, Equation 3.8 simply becomes

$$S = \sum_{l=1}^{2m} q_l^2 \tag{3.10}$$

An example of a super-ellipse fitting problem is shown in Figure 3.8, where the cloud of points are shown in red, a initial guess of super-ellipse as a green line, closest points in blue, and the distances from the points in cloud to the surface in black. The objective minimizes the sum of the lengths of the black lines.

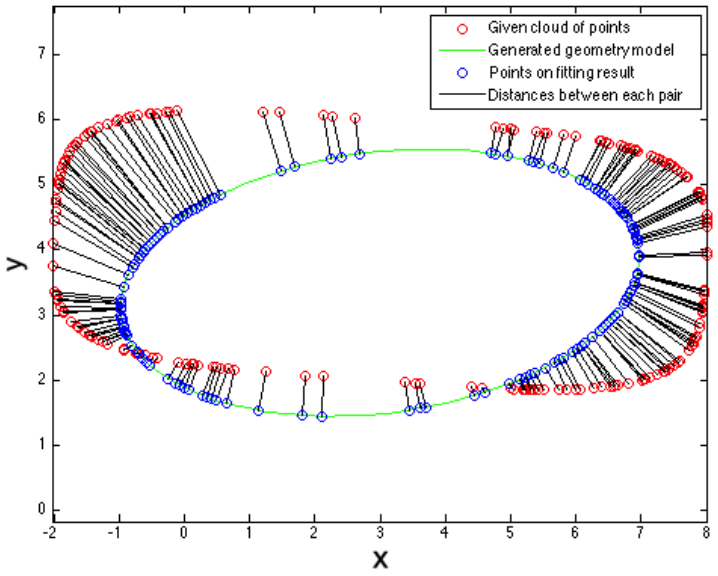


Figure 3.8: Cloud of points and sample super-ellipse.

3.1.3 Design Variables

As mentioned above, the optimizer certainly needs to be able to vary the elements of the design parameters $\vec{d} = (a, b, r, \delta x, \delta y, \theta)$ in order to achieve a good fit.

But since one does not know beforehand where on the surface of the super-ellipse each point in the cloud maps, m parametric coordinates, u_i, need to be added to the optimization's design variables; the elements of design parameters $\vec{d}$ and the parametric coordinates u_i will be automatically adjusted during the optimization process.

Thus, the actual design variables that can be changed by the optimizer in order to minimize S are

$$\vec{\beta} = \left[\vec{d}, u_1, \ldots, u_m\right] \tag{3.11}$$

Hence, there are $n + m$ values that the optimizer will vary.

3.2 Optimization Algorithms Selection

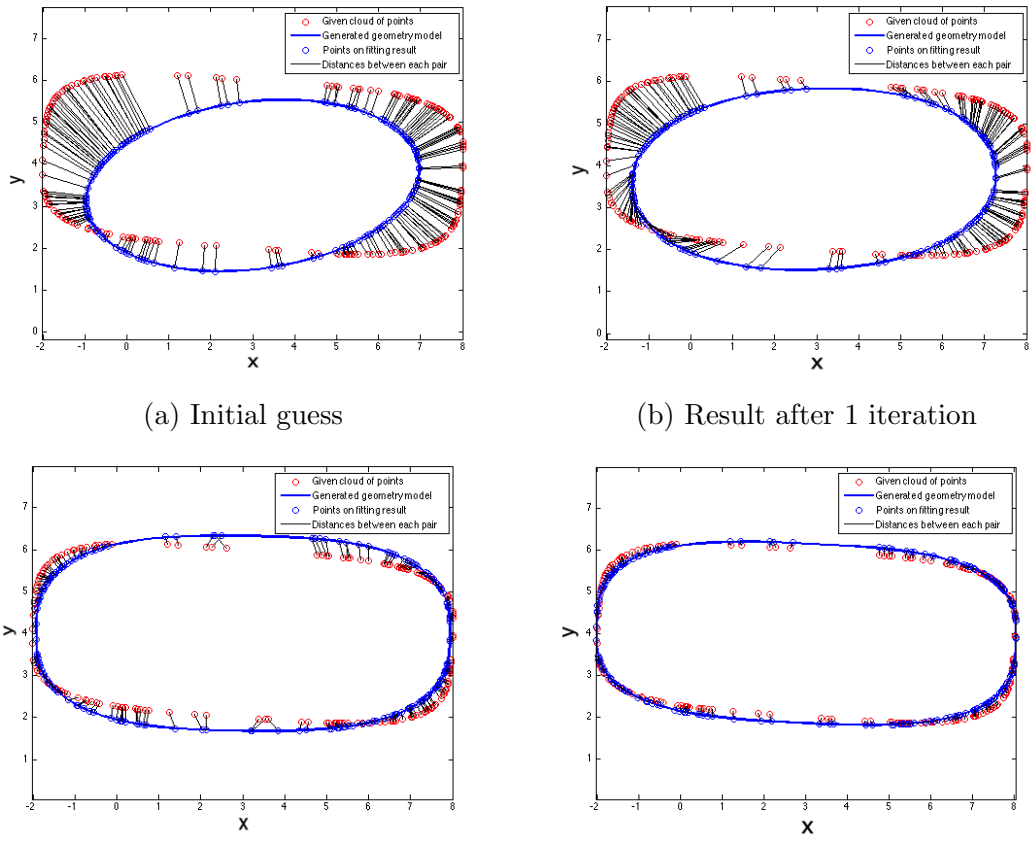
In this section, the various gradient-based optimization techniques that were described in Chapter 2 are applied to the current demonstration problem. Gradient-based techniques are used here since the equations necessary to computer the objective function, S, can all be differentiated analytically to produce the needed gradient and Hessian expressions.

3.2.1 Gradient Descent Algorithm

Application of the gradient decent method into the super-ellipse fitting problem gives the results that are shown in Figure 3.9. The maximum number of iterations is arbitrarily set to 20, which is consistent with the iteration limit used in the other optimization algorithms

(Newton's method, Levenberg-Marquardt method, and Improved Levenberg-Marquardt method).

As shown in the Figure 3.9, the resulting super-ellipse is not a very good fit to the cloud of points after 20 iterations.



(a) Initial guess

(b) Result after 1 iteration

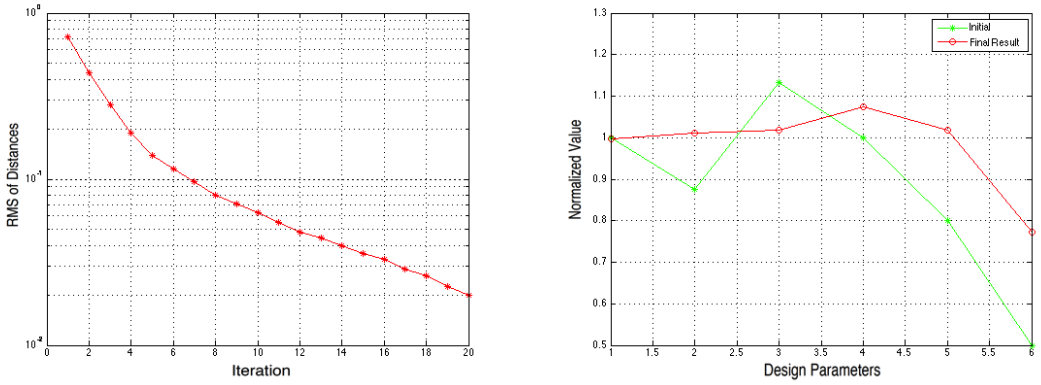
(c) Result after 10 iteration

(d) Result after 20 iteration

Figure 3.9: Results of super-ellipse generation based on gradient decent algorithm

The RMS of distances for each iteration is shown in Figure 3.10a. The history of RMS of distances shows that the optimization process is far from convergence as seen by the downward slope at the end of the plot. This means that if one runs more iterations, a better result might obtained. But this also means that it requires more time to get good results (as compared with the other techniques described below). The RMS is 0.3632 after 20 iterations.

Figure 3.10b shows the normalized design parameters d_N for both the initial guess and the final result. The normalized design parameter is defined as the design parameter divided by the value d_r (of the design parameter) that was used to generate the points in the cloud. ($d_{Ni} = d_i/d_r$) A good fit would have all the normalized design parameters equal to one, which is clearly not the case in the Figure.

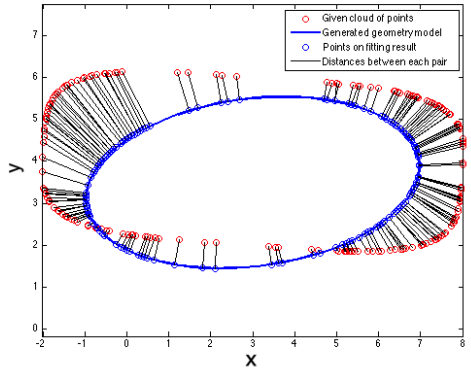


(a) RMS of distances (b) Normalized design parameters

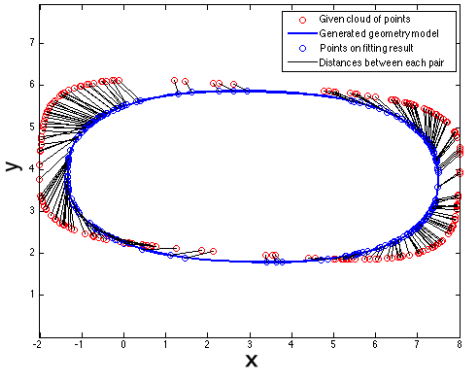
Figure 3.10: Fitting results analysis based on gradient decent algorithm

3.2.2 Newton's Algorithm

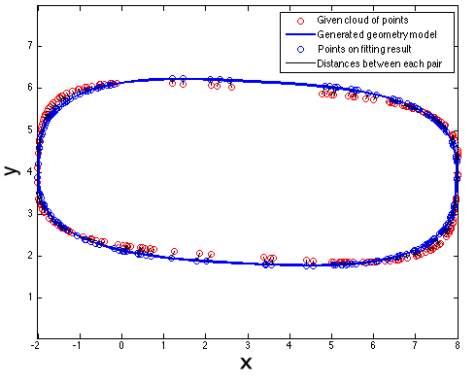
After applying the Newton's method into the super-ellipse fitting problem, the result can be shown in Figure 3.11. For the Newton's algorithm, the results of design parameters (a,b) are much better then the results from gradient decent method.



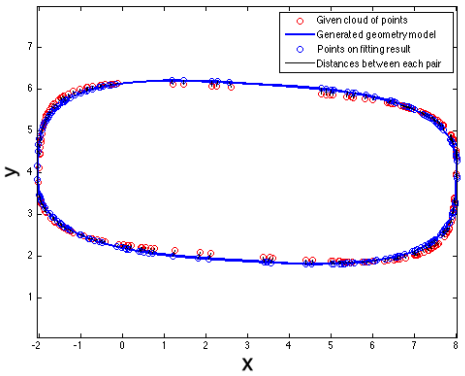
(a) Initial guess



(b) Result after 1 iteration



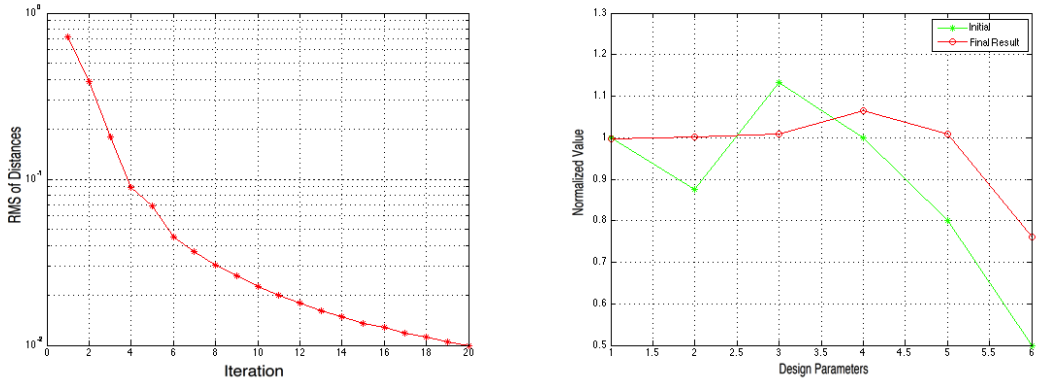
(c) Result after 10 iteration



(d) Finial result

Figure 3.11: Results of super-ellipse generation based on Newton's algorithm

The RMS of distances for each iteration is shown in Figure 3.12a. Like in the gradient descent method, the history of RMS of distances shows that the optimization process is also not converged after 20 iterations. The RMS is 0.01 after 20 iterations. Figure 3.12b also shows that the results of the Newton scheme are all far from 1, which indicates that the design parameters that were used to generate the cloud of points were not properly recovered.



(a) RMS of distances (b) Normalized design parameters

Figure 3.12: Fitting results analysis based on Newton's algorithm

3.2.3 Levenberg-Marquardt Algorithm

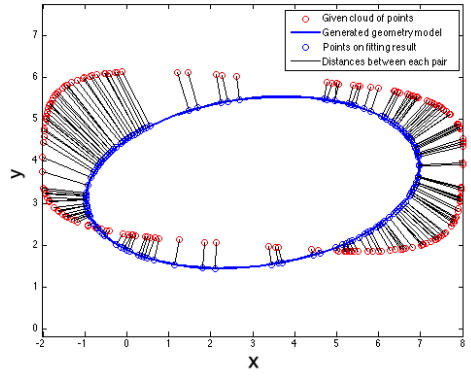
The iteration function of LM method is:

$$\vec{\delta} = -(J^T J + \lambda \cdot \text{diag}(J^T J))^{-1} \cdot J^T \vec{q} \tag{3.12}$$

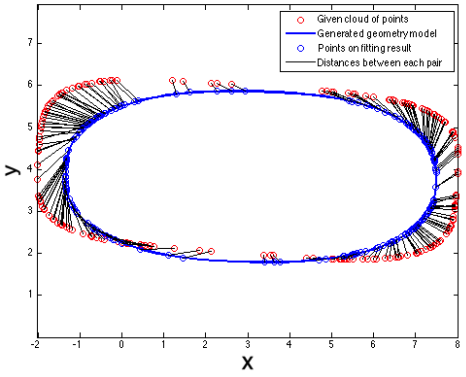
Note that $\vec{\delta}$ is $(m+n) \times 1$, $J^T \vec{q}$ is $(m+n) \times 1$, and $J^T J$ is $(m+n) \times (m+n)$.

The fitting results based on the LM algorithm will be shown below. For the LM algorithm, design parameters (a and b) are much better then the results from the gradient decent of Newton's methods.

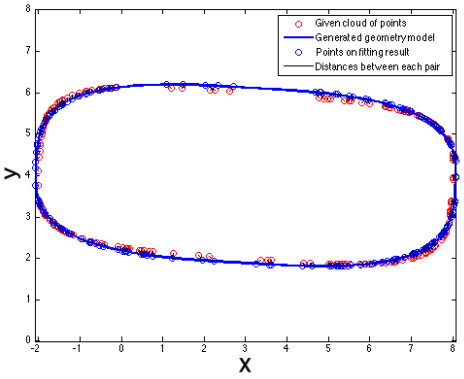
The results of design parameters during the optimization process are shown in Figure 3.13.



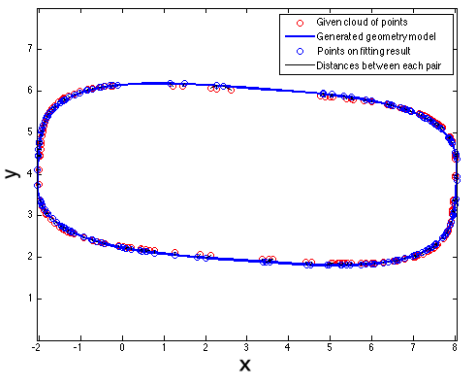
(a) Initial guess



(b) Result after 1 iteration



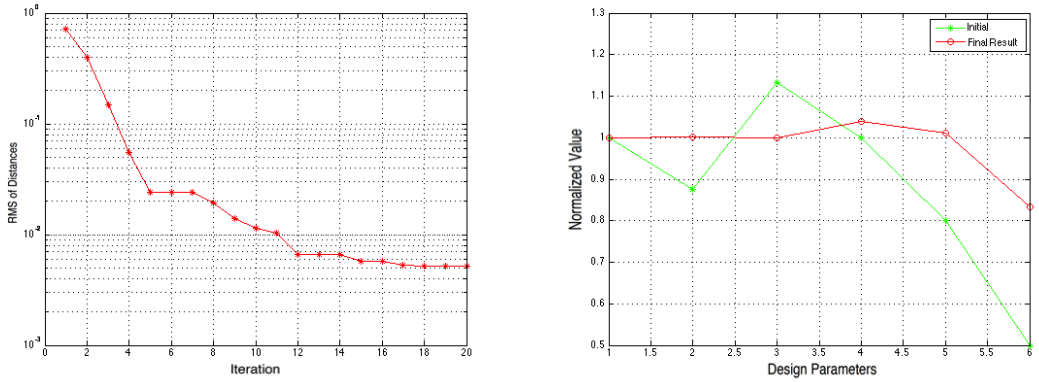
(c) Result after 10 iteration



(d) Finial result

Figure 3.13: Results of super-ellipse generation based on LM algorithm

The RMS of distances for each iteration is shown in Figure 3.14a, which do not change much during the last several iterations of the optimization process, indicating convergence. The RMS is 0.0052 after 20 iterations. From Figure 3.14b, the resulting normalized design parameters are closer to 1, as compared with the gradient decent or Newton's methods.

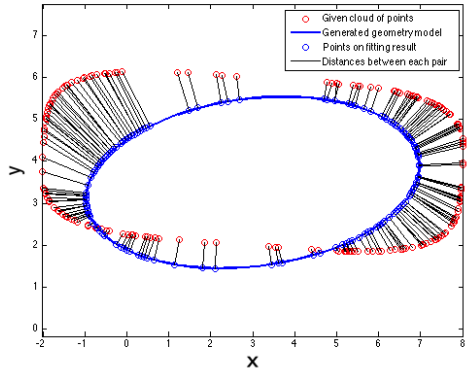


(a) RMS of distances (b) Normalized design parameters

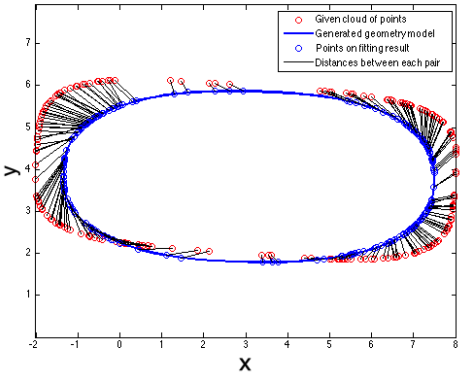
Figure 3.14: Fitting results analysis based on LM algorithm

3.2.4 Improved Levenberg-Marquardt Algorithm

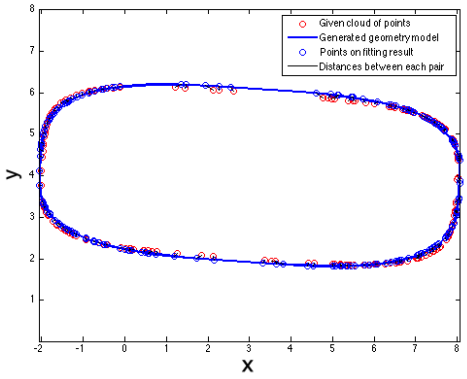
The fitting results based on the ILM algorithm are shown below. After applying the ILM algorithm, design parameters (a,b) are much closer to the correct values. The results of design parameters during the optimization process are shown in Figure 3.15.



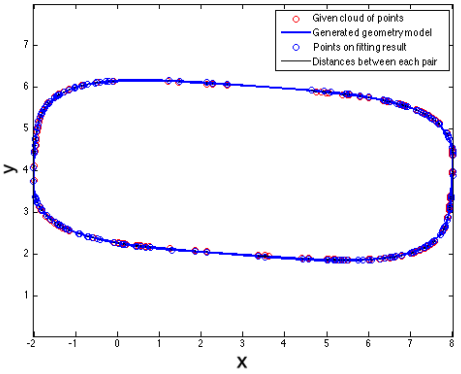
(a) Initial guess



(b) Result after 1 iteration



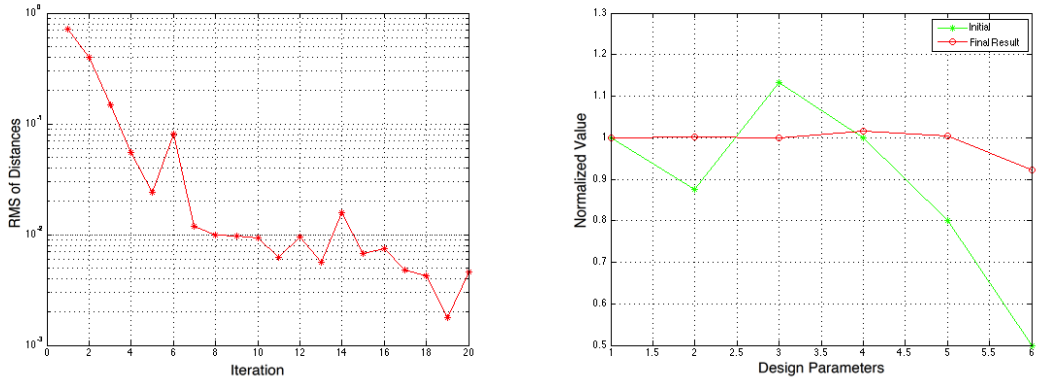
(c) Result after 10 iteration



(d) Finial result

Figure 3.15: Results of super-ellipse generation based on ILM algorithm

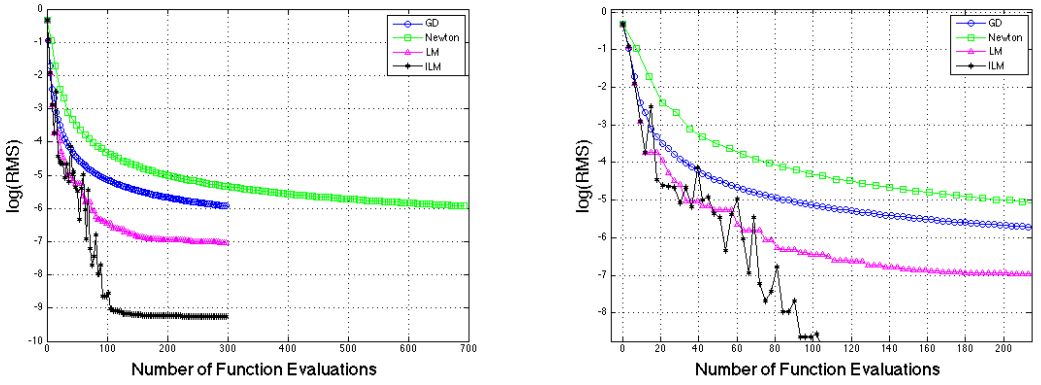
The RMS of distances for each iteration is shown in Figure 3.16a, which again indicates that ILM has converged for this problem in fewer then 20 iterations. The RMS is 0.0046 after 20 iterations. Again, Figure 3.16b shows that many of the design parameters have reached the values that were used to generate the cloud of points.



(a) RMS of distances (b) Normalized design parameters

Figure 3.16: Fitting results analysis based on ILM algorithm

For comparing the performance for applying different algorithms into geometry fitting problem, the RMS of distance history based on the number of function evaluations are shown in Figure 3.17. As shown in the figure, the ILM method can overcome the local minimum problem and the speed of convergence is fast.



(a) RMS of distances (b) Zoom in the figure

Figure 3.17: RMS of distances for applying different optimization algorithms

3.3 Sparse Technique for Matrix Calculation

Here, recall the iteration function of ILM and the function is shown as Equation 2.18

$$\vec{\delta} = -(J^T J + \lambda \cdot \mathrm{diag}(J^T J))^{-1} \cdot J^T \vec{q} \tag{3.13}$$

where J is the Jacobian matrix of objective function. J^T is the transpose of J. I is the identity matrix. The sparse techniques are applied to this function.

3.3.1 Generation of Sparse Jacobian Matrix

To improve the computational efficiency of the technique, one can see that the Jacobian matrix is very sparse, as shown in equation 3.14. For the first $2m \times n$ block, all the columns that correspond to the derivatives with respect to the design parameters, $\vec{d}$, are not generally zeros. But the second $2m \times m$ block of the Jacobian is comprised of two diagonal matrices, stacked one above the other, which correspond to the derivatives with respect to the parametric coordinate of that point, u_i.

$$J = \left[\begin{array}{ccc|ccc} \frac{\partial f_{x(1)}}{\partial d_1} & \cdots & \frac{\partial f_{x(1)}}{\partial d_n} & \frac{\partial f_{x(1)}}{\partial u_1} & 0 & \cdots & 0 \\ \frac{\partial f_{x(2)}}{\partial d_1} & \cdots & \frac{\partial f_{x(2)}}{\partial d_n} & 0 & \frac{\partial f_{x(2)}}{\partial u_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{x(m)}}{\partial d_1} & \cdots & \frac{\partial f_{x(m)}}{\partial d_n} & 0 & \cdots & \cdots & \frac{\partial f_{x(m)}}{\partial u_m} \\ \frac{\partial f_{y(1)}}{\partial d_1} & \cdots & \frac{\partial f_{y(1)}}{\partial d_n} & \frac{\partial f_{y(1)}}{\partial u_1} & 0 & \cdots & 0 \\ \frac{\partial f_{y(2)}}{\partial d_1} & \cdots & \frac{\partial f_{y(2)}}{\partial d_n} & 0 & \frac{\partial f_{y(2)}}{\partial u_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{y(m)}}{\partial d_1} & \cdots & \frac{\partial f_{y(m)}}{\partial d_n} & 0 & \cdots & \cdots & \frac{\partial f_{y(m)}}{\partial u_m} \end{array}\right] \tag{3.14}$$

Hence, sparse matrix methods can be employed to great advantage, both to reduce the required memory and to increase the computational speed. Additionally, since J only contains derivatives, if one computes the derivative analytically (for example, by using the techniques described in [121], additional time can be saved by not having to generate J via finite differences.

3.3.2 Arithmetic of Sparse Jacobian Matrix

After applying the sparse expression technique for J, one can reduce the memory required from $(2m \times n + 2m^2)$ to $(2m \times (n+1))$. Because $n \ll m$, the complexity of the J can be reduced from $O(m^2)$ to $O(m)$. The sparse J can be expressed as Equation 3.15

$$J = \left[\begin{array}{ccc|c} \frac{\partial f_{x(1)}}{\partial d_1} & \cdots & \frac{\partial f_{x(1)}}{\partial d_n} & \frac{\partial f_{x(1)}}{\partial u_1} \\ \frac{\partial f_{x(2)}}{\partial d_1} & \cdots & \frac{\partial f_{x(2)}}{\partial d_n} & \frac{\partial f_{x(2)}}{\partial u_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_{x(m)}}{\partial d_1} & \cdots & \frac{\partial f_{x(m)}}{\partial d_n} & \frac{\partial f_{x(m)}}{\partial u_m} \\ \frac{\partial f_{y(1)}}{\partial d_1} & \cdots & \frac{\partial f_{y(1)}}{\partial d_n} & \frac{\partial f_{y(1)}}{\partial u_1} \\ \frac{\partial f_{y(2)}}{\partial d_1} & \cdots & \frac{\partial f_{y(2)}}{\partial d_n} & \frac{\partial f_{y(2)}}{\partial u_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_{y(m)}}{\partial d_1} & \cdots & \frac{\partial f_{y(m)}}{\partial d_n} & \frac{\partial f_{y(m)}}{\partial u_m} \end{array}\right] \tag{3.15}$$

As in the iteration function $\vec{\delta} = -(J^T J + \lambda \cdot \mathrm{diag}(J^T J))^{-1} \cdot J^T \vec{q}$, the multiplication for sparse matrix need to be developed for $J^T J$ and $J^T \vec{q}$.

For the computation of $H = J^T J$, the structure of H is shown in Figure 3.18. A is an $n \times n$ block, and its multiplication algorithm is the same as the general matrix multiplication. Because $B = C^T$, there only is a need to calculate once for block B or C. So, for B, the

formula for calculating each element in it is as Equation 3.16

$$B_{(i,j)} = \frac{\partial f_{x(i)}}{\partial u_{(i)}} \cdot \frac{\partial f_{x(i)}}{\partial d_{(j)}} + \frac{\partial f_{y(i)}}{\partial u_{(i)}} \cdot \frac{\partial f_{y(i)}}{\partial d_{(j)}} \tag{3.16}$$

For the block D, it is a diagonal matrix and the formula for calculating D is as Equation 3.17

$$D_{(i,i)} = \left(\frac{\partial f_{x(i)}}{\partial u_{(i)}}\right)^2 + \left(\frac{\partial f_{y(i)}}{\partial u_{(i)}}\right)^2 \tag{3.17}$$

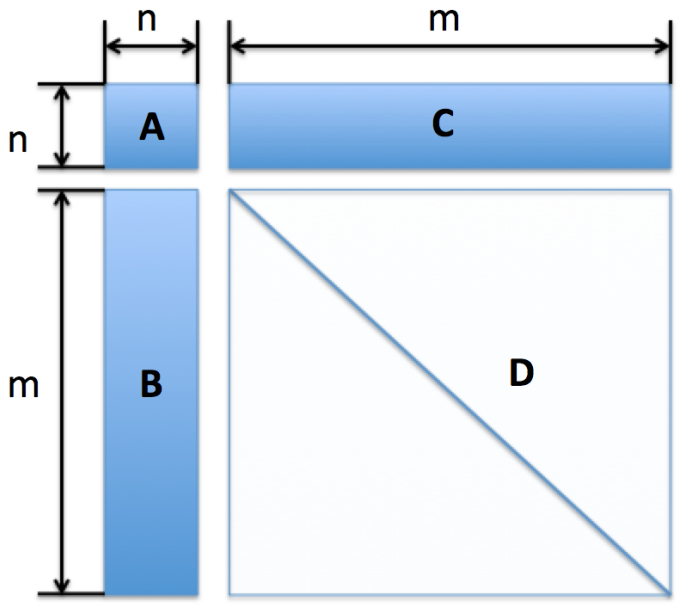


Figure 3.18: Hessian matrix structure

Consider now the computation of $\vec{g} = J^T\vec{q}$. After the multiplication, the $\vec{g}$ is a vector and its length is $n+m$. For the first n elements, the general matrix multiplication algorithm is used. For the lest m elements, the formula can be calculated as Equation 3.18.

$$g_{(n+i)} = \frac{\partial f_{x(i)}}{\partial u_{(i)}} \cdot q_{(i)} + \frac{\partial f_{y(i)}}{\partial u_{(i)}} \cdot q_{(i+m)} \tag{3.18}$$

3.3.3 Solving Sparse Matrix System

In order to solve the sparse linear system, the conjugate gradient method is used. This is a very attractive technique for the numerical solution of sparse, symmetric, positive-definite matrices. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods. See [122] for details of the algorithm used here.

Chapter 4

Fitting a General Model to a Cloud of Points

In Chapter 3, the Levenberg-Marquardt optimization method was applied to the problem of fitting a simplified model to a given cloud of points. The optimization technique simultaneously varied the design parameters, $\vec{d}$ and the parametric coordinates, u_i associated with each point in the cloud. In that Chapter, it was assumed that one had a good guess for the initial values of the parametric coordinates, u_i.

This Chapter focuses on generalizing the results of Chapter 3 in three important ways. In the first, an initialization technique that generates a reasonable set of initial parametric coordinates, u_i, is described. The second generalization involves configurations that have more than one component; in these cases one needs to determine the part of the configuration to which each point in the cloud is associated. This leads directly to a new classification technique. Finally, the basic fitting technique is extended to three-dimensional configurations. The Chapter concludes with a overview of the entire fitting process, including classification and initialization.

4.1 (Re)Initialization Technique

In this section, a rational initialization technique is developed for obtaining the initial parametric coordinates u_i that are closest to each point in the cloud. In what follows, it is assumed that the user has provided a guess for the initial design parameters $\vec{d}$ that generate a configuration that resembles the cloud of points, but is not in general a "good" fit. In other words, if the cloud of points represent a transport aircraft, the user's guess for the design parameters must produce an aircraft of approximately the same size and arrangement as the points in the cloud.

4.1.1 Basic Idea

Given a set of design parameters, $\vec{d}$, one can represent a configuration discretely with a set of line segments (in two dimensions) or triangular or quadrilateral patches (in three dimensions). These segments or patches, which are bounded by vertices, are used to graphically represent the configuration.

The basic idea for the initialization of the u_i is to find the vertex in the discrete representation that is closest to point (x_i, y_i) in the cloud and then use that vertex's parametric coordinate as the initial guess of the (cloud's) point u_i. This process is shown in Figure 4.1, where the points in the cloud are shown as red symbols. The initial discrete representation is shown as the green line segments and its vertices are shown as blue symbols. The correspondence between each (red) point in the cloud and the closest (blue) discrete vertex is shown as the black lines.

The process of making these correspondences consists of nested loops over the points in the cloud and over the vertices in the discrete representation. During this process, the

shortest distance from each point in the cloud to any vertex, e_i, is squared and all are summed to generate the initial value of the objective function, $S = \sum_i e_i^2$.

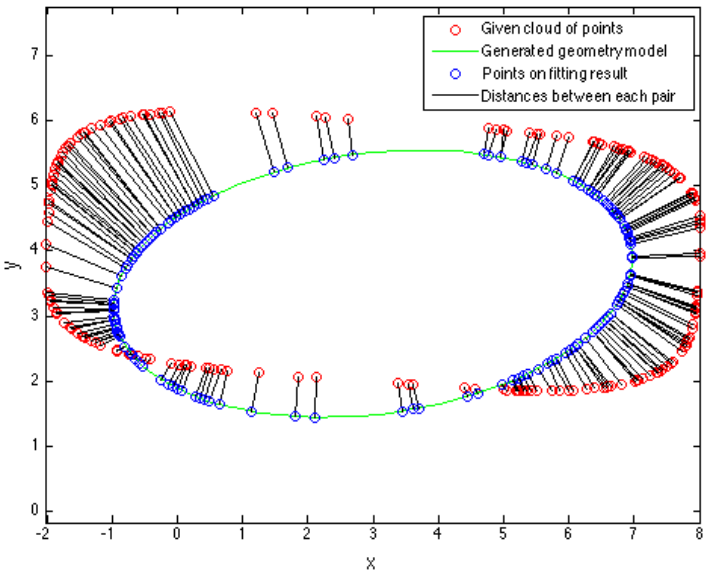


Figure 4.1: Demonstration of initialization technique

Given the initial guesses of the u_i that are shown in Figure 4.1, one can apply the LM technique (described in Chapter 3) to produce the final optimized fit shown in Figure 4.2.

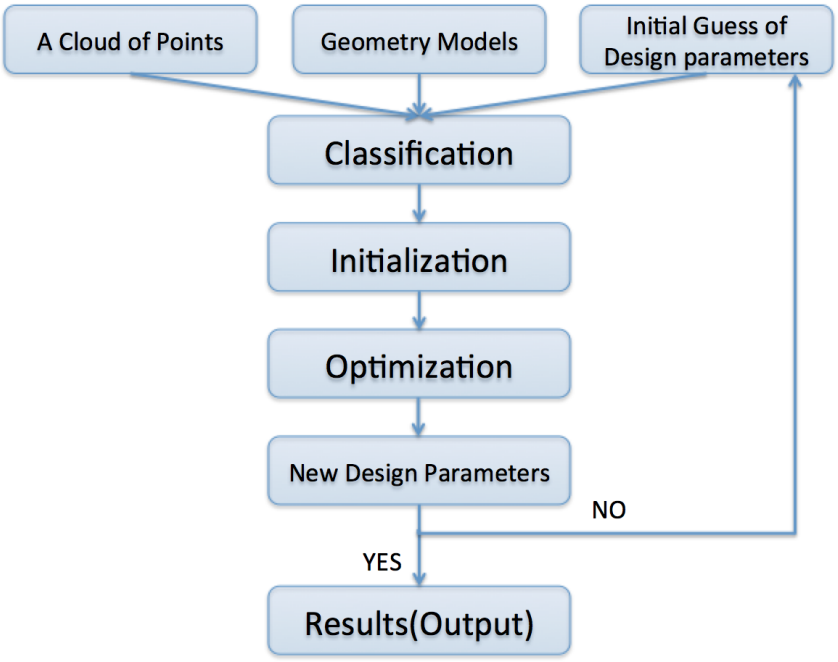


Figure 4.2: Flowchart for fitting parametric geometry model from a cloud of points

4.1.2 Dealing with Bad Initial Guesses for $\vec{d}$

The above process generates good initial guesses for the u_i if the initial design parameters, $\vec{d}$, produces a configuration that is a reasonable approximation to the points in the cloud. But this is not always possible, especially for very complex three-dimensional configurations.

Consider the initial configuration and cloud of points shown in Figure 4.3. Note that for three points in the cloud, the "shortest" distance between the point and the vertices in the discrete representation erroneously makes a correspondence to the wrong side of the super-ellipse. Executing the LM optimizer from this initial guess yields the "final" results shown in Figure 4.4. Note that points that were badly initialized remain on the wrong side of the super-ellipse. The reason that this happens is that any adjustment of the u_i (temporarily) increases the distance to the point in the cloud.

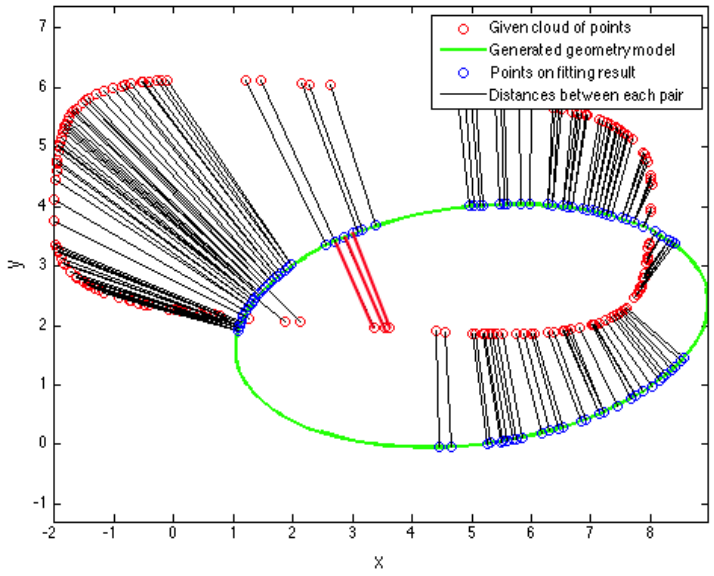
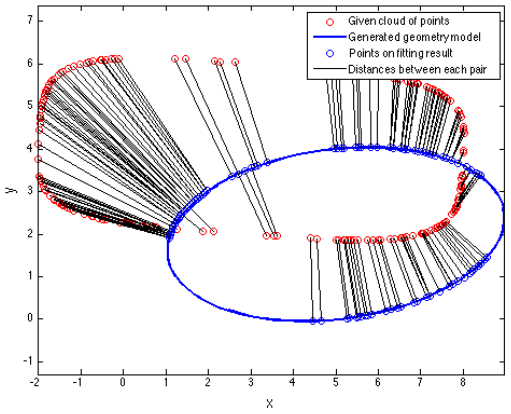
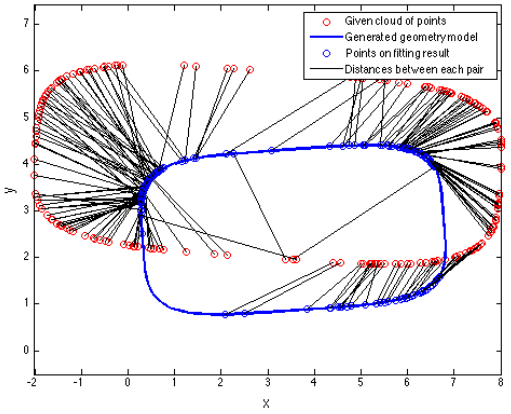


Figure 4.3: Initialization based on bad initial design parameters

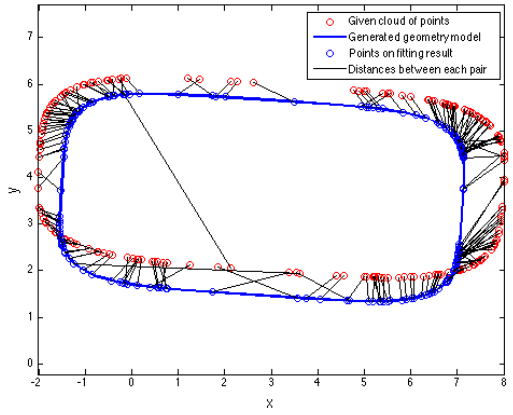
Here, the original Levenberg-Marquardt method is used as the optimization algorithm. The reason for not using ILM method is that the ILM method can overcome the local minimum problem somehow. For observing the stuck at local minimum problem due to the bad initial design parameters, the monotonic optimization method (LM) is chosen. For making sure that there is not a problem from un-converged, the maximum number of iterations is set as 100. The fitting results of design parameters during the optimization process are shown in Figure 4.4.



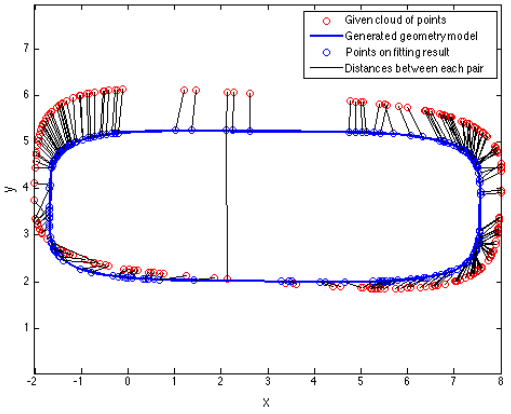
(a) Initial guess



(b) Result after 1 iteration

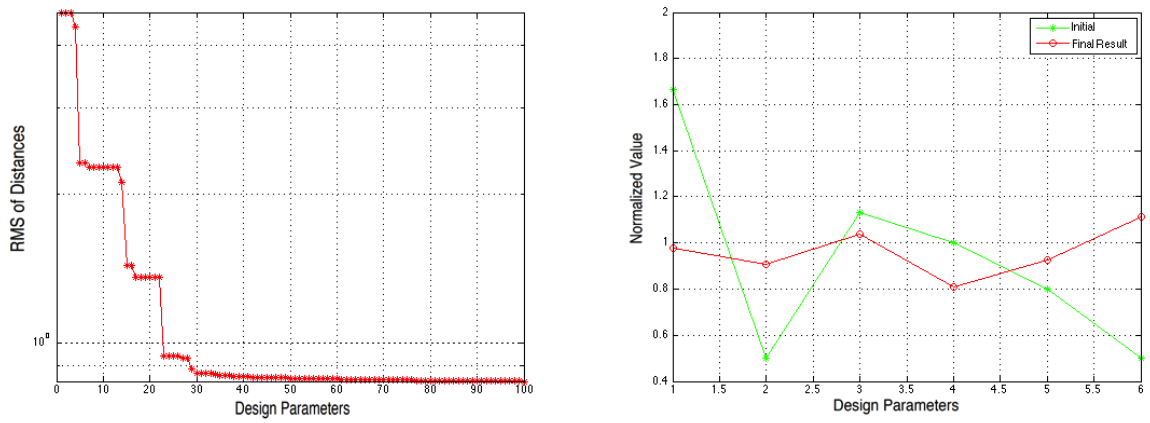


(c) Result after 10 iteration



(d) Finial result

Figure 4.4: Results of super-ellipse generation start from bad initial guess (LM)

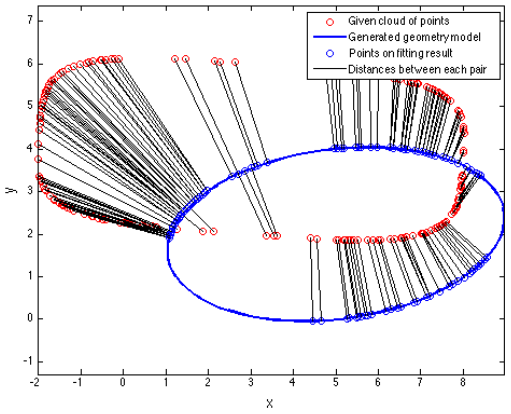


(a) RMS of distances

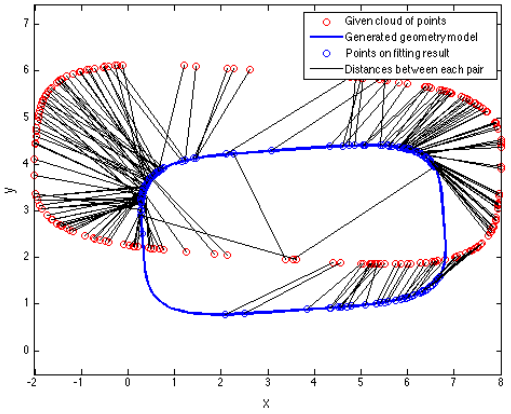
(b) Normalized design parameters

Figure 4.5: Fitting results analysis for the bad initial guess problem (LM)

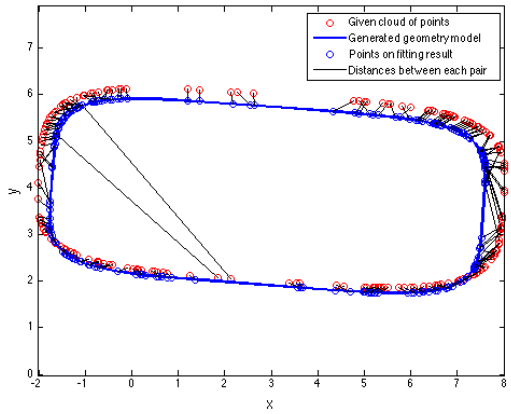
One method for overcoming this problem is to apply the Improved Levenberg-Marquardt (ILM) algorithm as the optimization method. Recall that ILM allows the optimizer to sometimes take "uphill" steps, especially at the early stage of the optimization. Figure 4.6d shows that ILM preforms better than LM, even when both techniques are limited to 100 iterations. The better performance of ILM over LM is indeed good, but also fortuitous, because there is no guarantee that ILM will always be able to overcome bad initial u_i values.



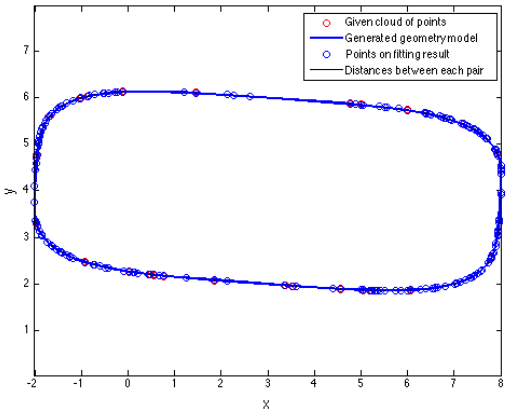
(a) Initial guess



(b) Result after 1 iteration

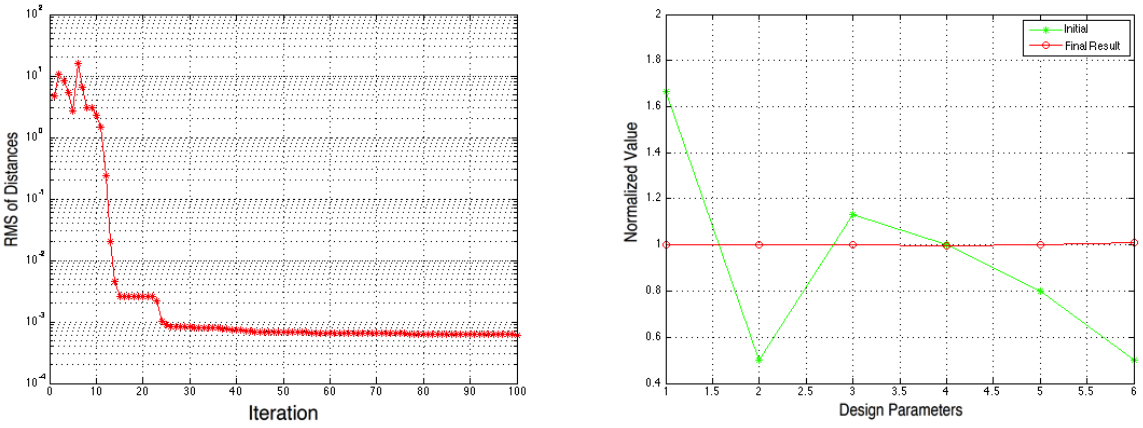


(c) Result after 10 iteration



(d) Finial result

Figure 4.6: Results of super-ellipse generation start from bad initial guess (ILM)



(a) RMS of distances

(b) Normalized design parameters

Figure 4.7: Fitting results analysis for the bad initial guess problem (ILM)

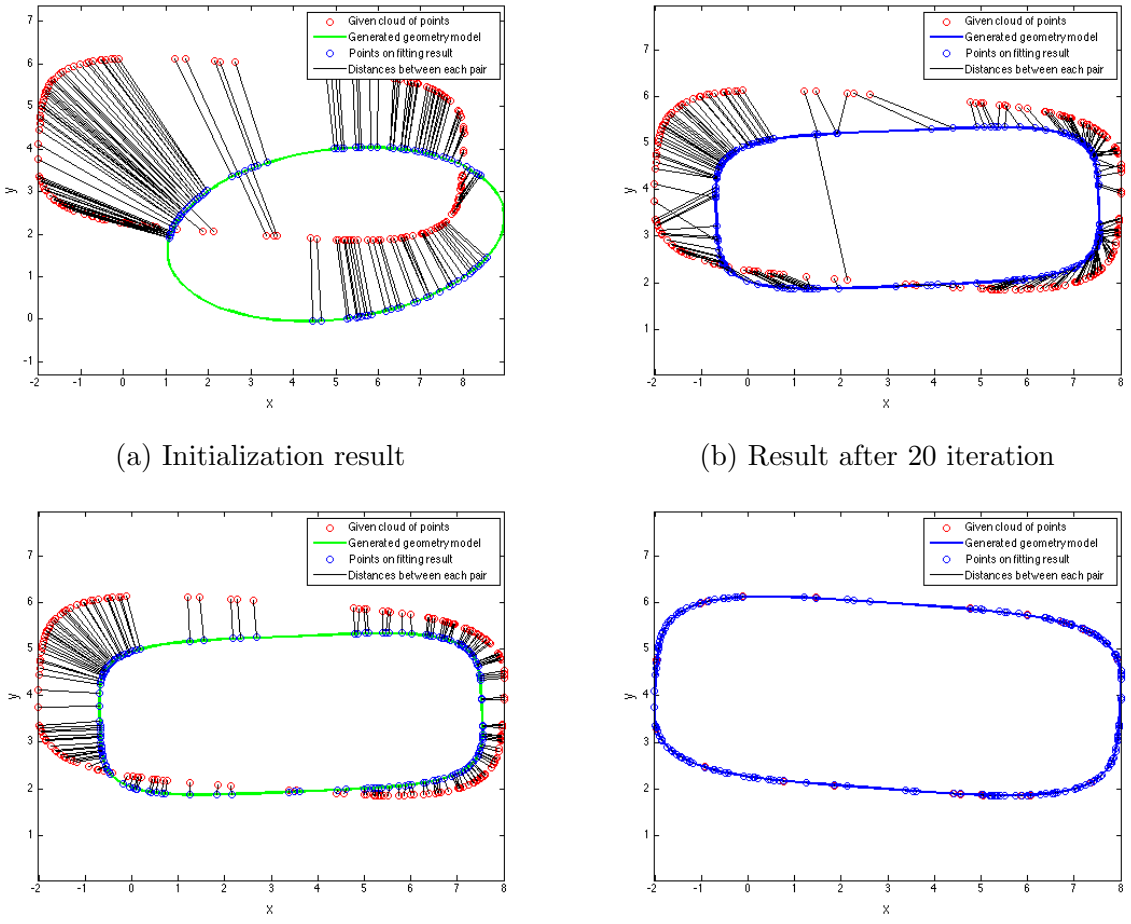
4.1.3 Reinitialization

Although the ILM algorithm can solve this kind of problem in super-ellipse case, there is no guarantee that the ILM can solve this kind of problem during the more complicated geometry fitting process. Moreover, for obtaining the good design parameters, one needs to run many ILM iterations that are time consuming. So, a more general method for solving this problem is introduced in this section.

To solve this problem, one can simply reinitialize the u_i, while keeping the best design parameters $\vec{d}$. The reinitialization technique generates a new sequence of u_i based on the closest distances to the vertices associated with the latest $\vec{d}$. It is expected that this will overcome the local minimum problem.

For testing the performance of reinitialization technique, the original LM algorithm is used as optimization technique. The whole process is shown in Figure 4.8. The initializing result is shown in Figure 4.8a and the optimization result after the first 20 iterations

is shown in Figure 4.8b. As shown in the figure, the process is stopped by reaching the maximum number of iterations. Then, keeping the new design parameters that are obtained by the first 20 iterations, the reinitializing process is executed, producing the result shown in Figure 4.8c. Finally, executing the LM optimizer with the better guess for the u_i produces the result shown in Figure 4.8d.



(a) Initialization result

(b) Result after 20 iteration

(c) Reinitialization result

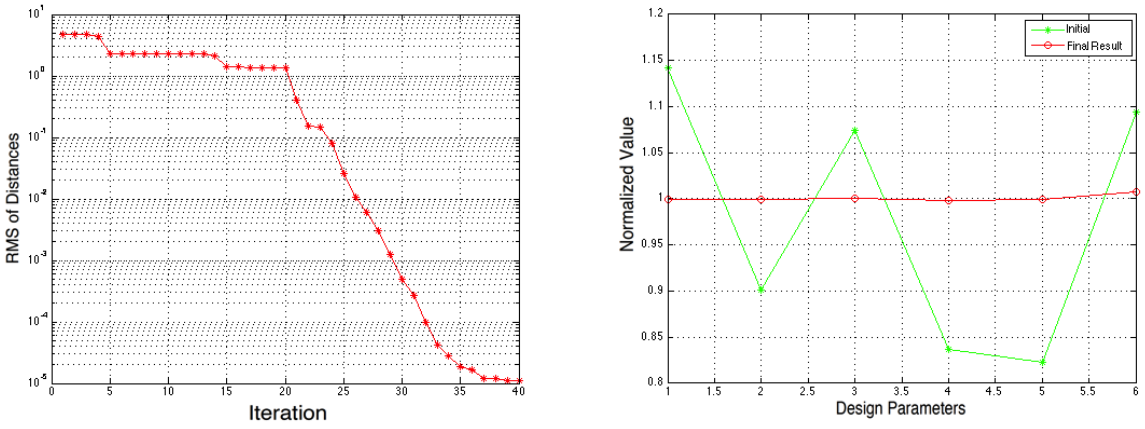
(d) Finial result

Figure 4.8: Results of super-ellipse generation using reinitialization technique

Here, a new item, "cycle" is defined. One *cycle* means the process of one *initialization* followed by one *optimization*. In each optimization process (in each cycle), there are at most `IterMax` iterations, which is set at the beginning of fitting process by the user.

Similarly, the user selects `CycleMax` as the maximum number of allowable cycles. Care must be taken in selecting `IterMax` and `CycleMax`. One the one hand, if the generation process involves too many cycles, most running time will be spent on the reinitialization but not on iterations for searching the correct design parameters. On the other hand, if the `IterMax` is too large, most running time will be spent on searching the design parameters near the local minimum area. From experience, it has been found that `CycleMax` is set as 5-10 based on complexity of the geometry configuration, and the `IterMax` is set as 20-30.

As shown in Figure 4.9a, during the super-ellipse fitting process, the RMS of distances will continue being reduced after 20 iterations, since the reinitializing process is taken. The total number of iterations is 40 now. It is much less compared with 100 iterations when there is no reinitialization technique. The normalized design parameters are equal to 1 as shown in Figure 4.9b.



(a) RMS of distances

(b) Normalized design parameters

Figure 4.9: Fitting results analysis after using reinitialization technique

4.1.4 Application to Fitting an Airfoil

Fitting an airfoil to a cloud of points is more difficult than the super-ellipse fitting, owing to the airfoil's thinness. This raises the probability that the wrong side of the airfoil will be chosen during the initialization.

For this case, a NACA 4-digit airfoil will be used [123]. The thickness distribution for a NACA 4-digit airfoil is given by

$$y_t = 5t \cdot c \cdot \left[0.2969\sqrt{\frac{x_i}{c}} + (-0.1260)\frac{x_i}{c} + (-0.3516)\frac{{x_i}^2}{c} + 0.2843\frac{{x_i}^3}{c} + (-0.1015)\frac{{x_i}^4}{c}\right] \tag{4.1}$$

where, x_i is the position along the chord line from 0 to c. y_t is the thickness at x_i location. c is the length of chord. t is the maximum thickness as a fraction of the chord.

Mean camber line function of NACA 4-digit airfoil is

$$y_c = \begin{cases} m \cdot \frac{x_i}{p^2} \cdot (2p - \frac{x_i}{c}) \\ m \cdot \frac{c - x_i}{1 - p^2} \cdot (1 + \frac{x_i}{c} - 2p) \end{cases} \tag{4.2}$$

where, the y_c is the y coordinate of the camber line at each location x_i. m is the maximum camber as a fraction of the chord and p is the location of maximum camber as a fraction of chord.

Coordinates of upper and lower of airfoil surfaces can be generated by "adding" the thickness to the camber line. This gives

$$\begin{cases} x_U = x_i - y_t \cdot sin\beta, \; x_L = x_i + y_t \cdot sin\theta_c \\ y_U = y_c + y_t \cdot cos\beta, \; y_L = y_c - y_t \cdot cos\theta_c \end{cases} \tag{4.3}$$

where,

$$\theta_c = \arctan\left(\frac{dy_c}{dx}\right), \tag{4.4}$$

$$\frac{dy_c}{dx} = \begin{cases} \dfrac{2m_{NACA}}{p^2}\left(p - \dfrac{x}{c}\right), & 0 \le x \le pc \\ \dfrac{2m_{NACA}}{(1-p)^2}\left(p - \dfrac{x}{c}\right), & pc \le x \le c \end{cases} \tag{4.5}$$

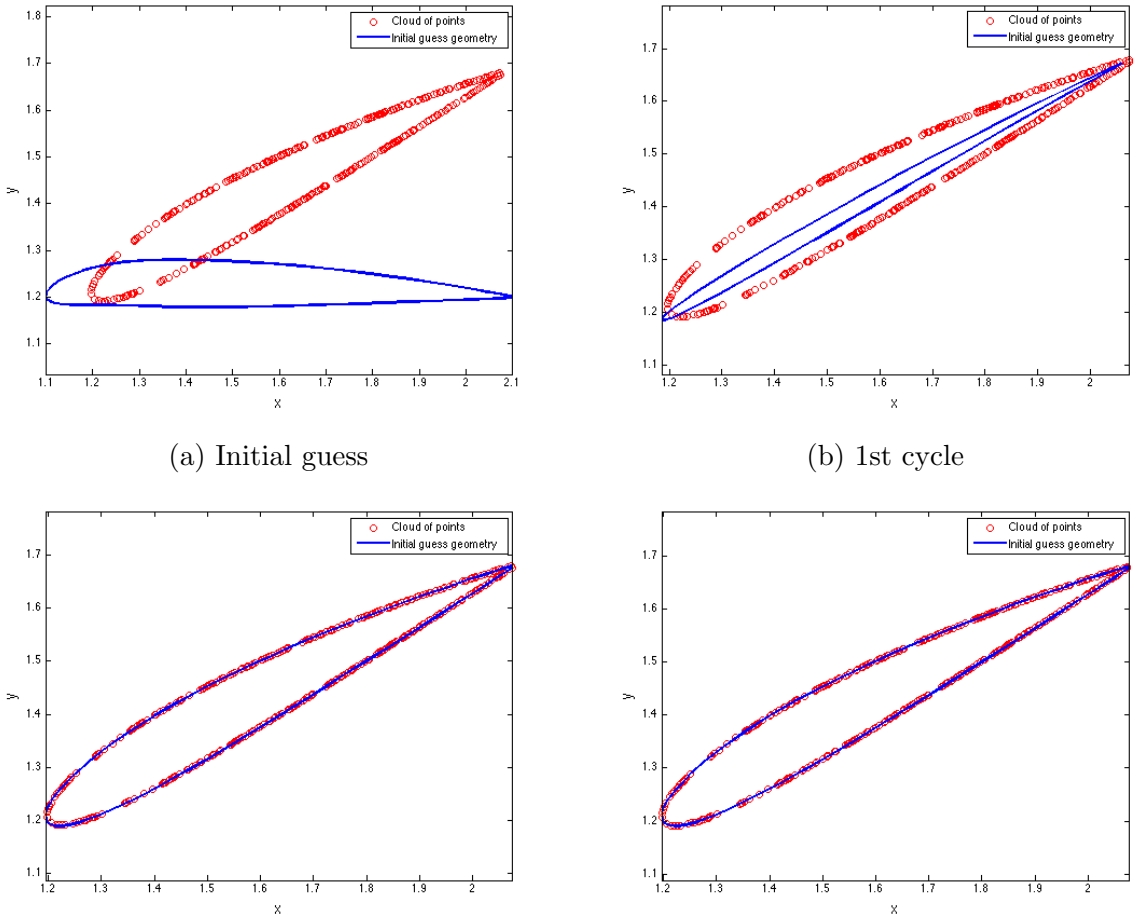
The above equations assume that the airfoil is generated so as to aligned with the x axis. In order to put the airfoil at an arbitrary angle, the homogeneous coordinates technique is applied in to $[x_L, y_L]$ as same as $[x_U, y_U]$, or

$$\begin{bmatrix} X_L \\ Y_L \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_L \\ yL \\ 1 \end{bmatrix} \tag{4.6}$$

For testing the robustness of the reinitialization algorithm, the similar process can be applied to a model that consists of a single NACA airfoil, shown in Figure 4.10. The red points are the points in cloud and the blue lines are the parametric model. Here, the maximum number of iterations is set as 10 for each cycle. For the airfoil problem, the correct fitting also can be obtained in only 2 cycles. In this case, chord length is fixed at one. The design parameters chosen for the NACA airfoil generation are the maximum camber m, location of maximum camber p, maximum thickness t, location $(\delta x, \delta y)$, and orientation angle θ.

The Figure 4.11 shows the RMS of distances and normalized design parameters. There is also mis-associated some points to the wrong side of configuration after the first 10

iterations, as in Figure 4.11a. But after reinitialization, this problem is overcame. The normalized of result design parameters are almost equal to 1 in Figure 4.11b.



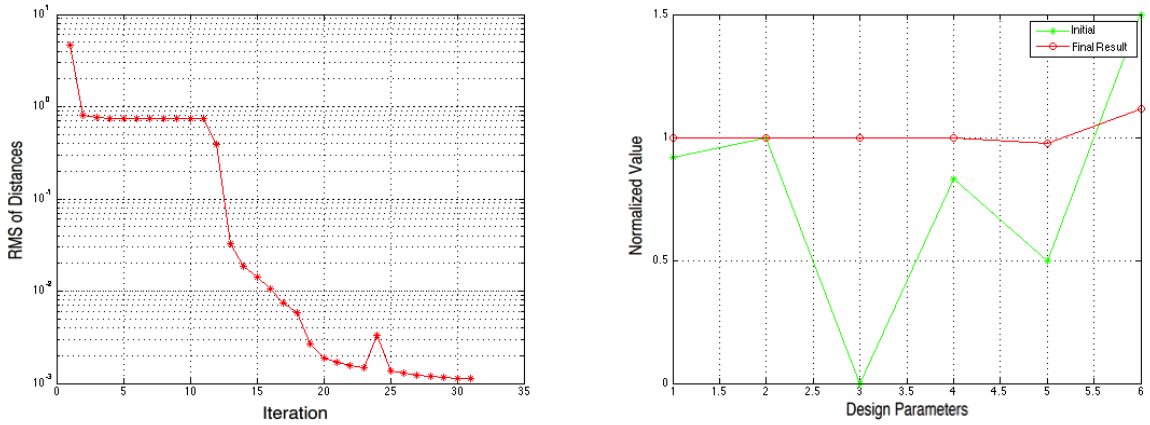
(a) Initial guess

(b) 1st cycle

(c) 2nd cycle

(d) 3rd cycle

Figure 4.10: Generation of NACA airfoil parametric model from a cloud of points



(a) RMS of distances

(b) Normalized design parameters

Figure 4.11: Airfoil fitting results analysis

Therefore, one expects that combining the ILM algorithm and reinitialization technique together, the whole method will be more robust and efficient. From now on, the later sections will use the new algorithm (ILM + reinitialization) for generation of parametric models from cloud of points.

4.2 Modified Objective Function

There is also another problem which can arise with models that will be discussed in this section. The super-ellipse is also used as a demonstration case. This kind of problem can be solved by modifying the objective function in general.

4.2.1 Bad Initial Guess Problem

The problem can arise with a model such as this. It occurs when the initial guess is much bigger than the cloud of point, as shown in Figure 4.12. The red points are the points in

the cloud. The green line is initial guess of super-ellipse. The black lines are the closest distances from the points in cloud to the geometry configuration.

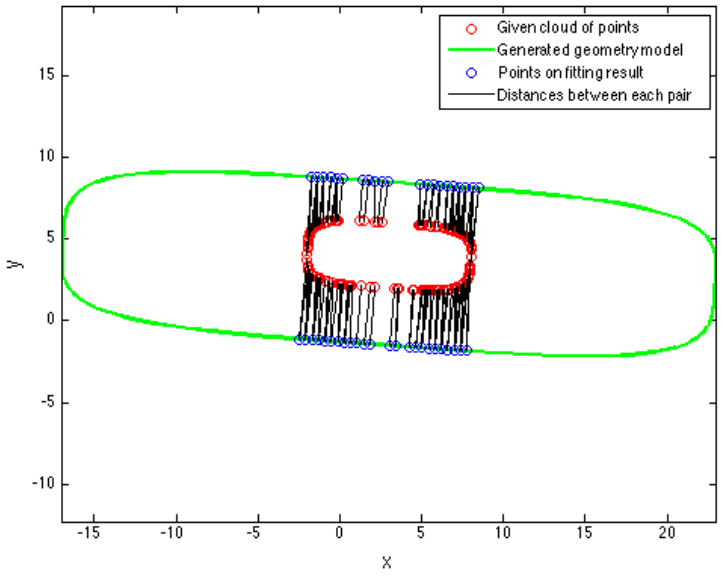
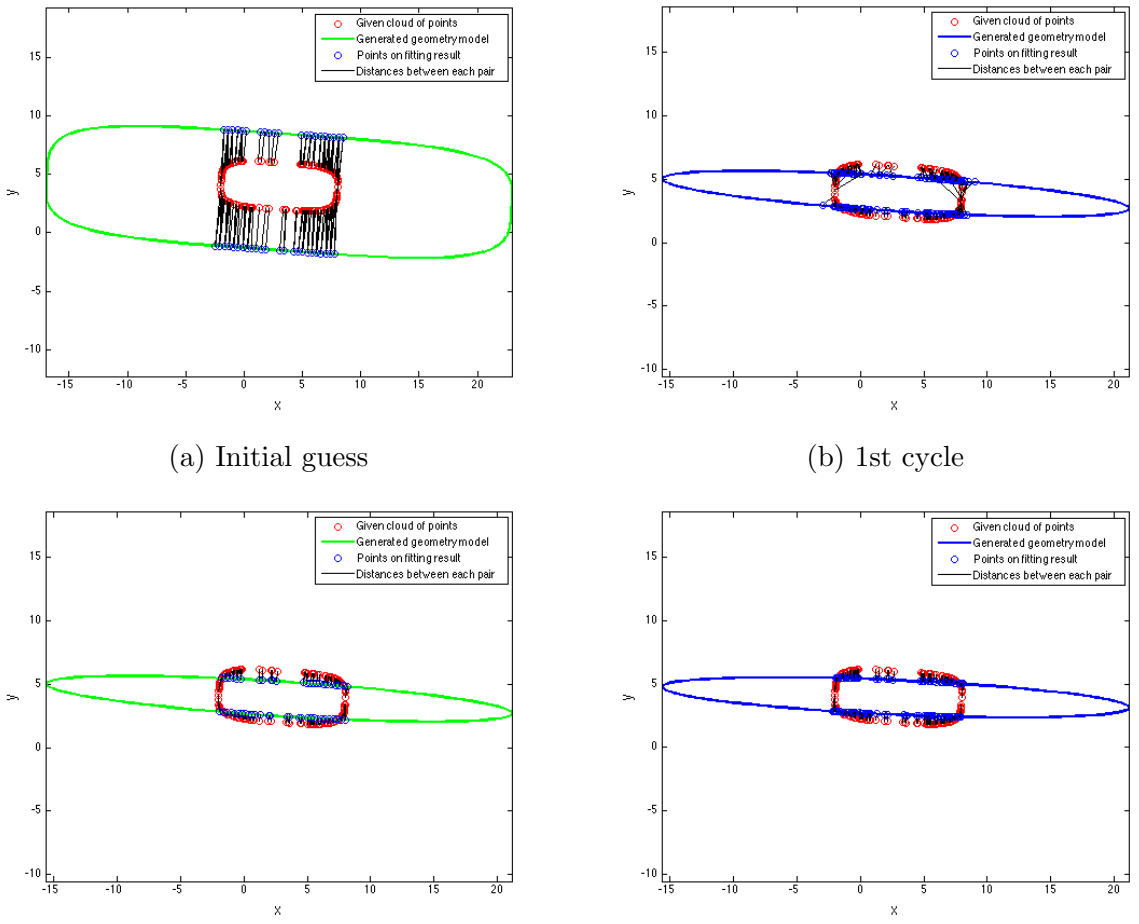


Figure 4.12: Initialization based on too large initial guess

Here, there is a significant part of the model that is not associated with any points in the cloud. In general, the optimizer will not be able to detect and fix this since all the points in the cloud may be close to (a portion of) the configuration. To solve this, a penalty function that penalizes configurations that are too big can be added to the objective function. The penalty can be related to the surface area or volume of the configuration. When this is done, one must take care to gradually reduce the weight of the penalty so that the penalty for the final fit vanishes. At this time, a general way of doing this has not been found that does not require the use of user-specified values for the weight of the penalty term.

This problem is tested by ILM algorithm combined with reinitialization technique. There are 2 cycles and 20 iterations for each cycle. The generation process and the result are shown in Figure 4.13 4.14.



(a) Initial guess

(b) 1st cycle

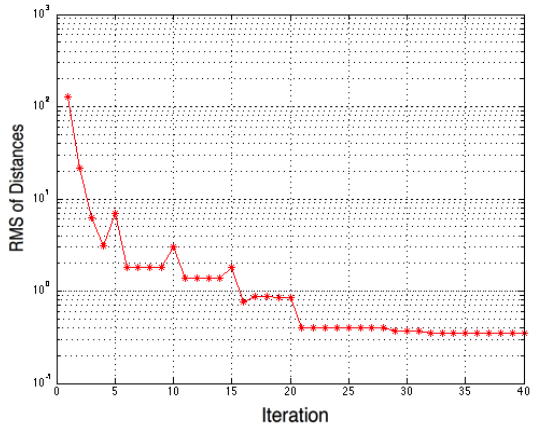
(c) Reinitialization result

(d) Final result

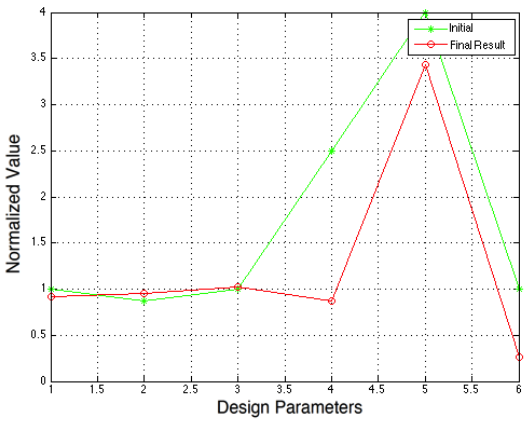
Figure 4.13: Results of super-ellipse generation start from too large initial guess (ILM)

As shown in the Figure 4.13, the accurate design parameters cannot be obtained even when the ILM algorithm applied after running 2 cycles. The RMS is reduced at first 20 iterations, but not changed more later in Figure 4.14a. The normalized length of super-ellipse (b) was not moving to 1 in Figure 4.14b. This is because there is no parametric coordinates of the model that is associated with the left and right sides points in the cloud.

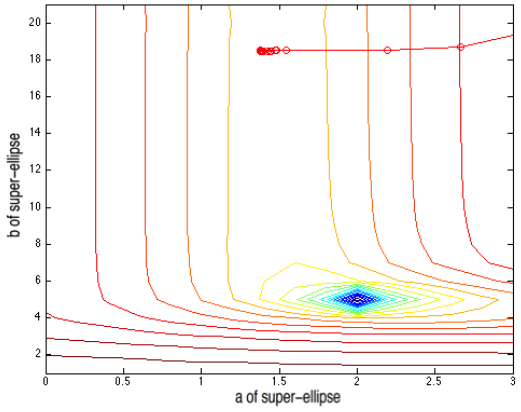
For observing the gradient change easily, log(objective function) is used in contours and 2D plots. In the Figure 4.14d, the log of objective function value has a flat area. No matter increased or decreased (a lot) the length b, the objective function will not be changed. Therefore, the optimizer is stuck at this area. The same explanation can be applied on Figure 4.14c; there is no contour lines along b direction. This means that the gradient is equal to zero at this area, and the optimizer does not know which direction need to move. For solving this problem in two cycles, in other words, more efficiently, one can improve the slop of the plot in Figure 4.14d or improve the gradient in Figure 4.14c. After doing this, the optimizer can know the moving direction. This can be achieved by adding a penalty term into the objective function.



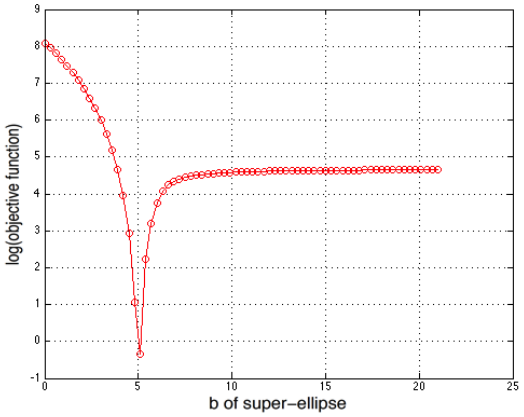
(a) RMS of distances



(b) Normalized design parameters



(c) Trajectory of design parameters a and b



(d) log(objective value) changed with b

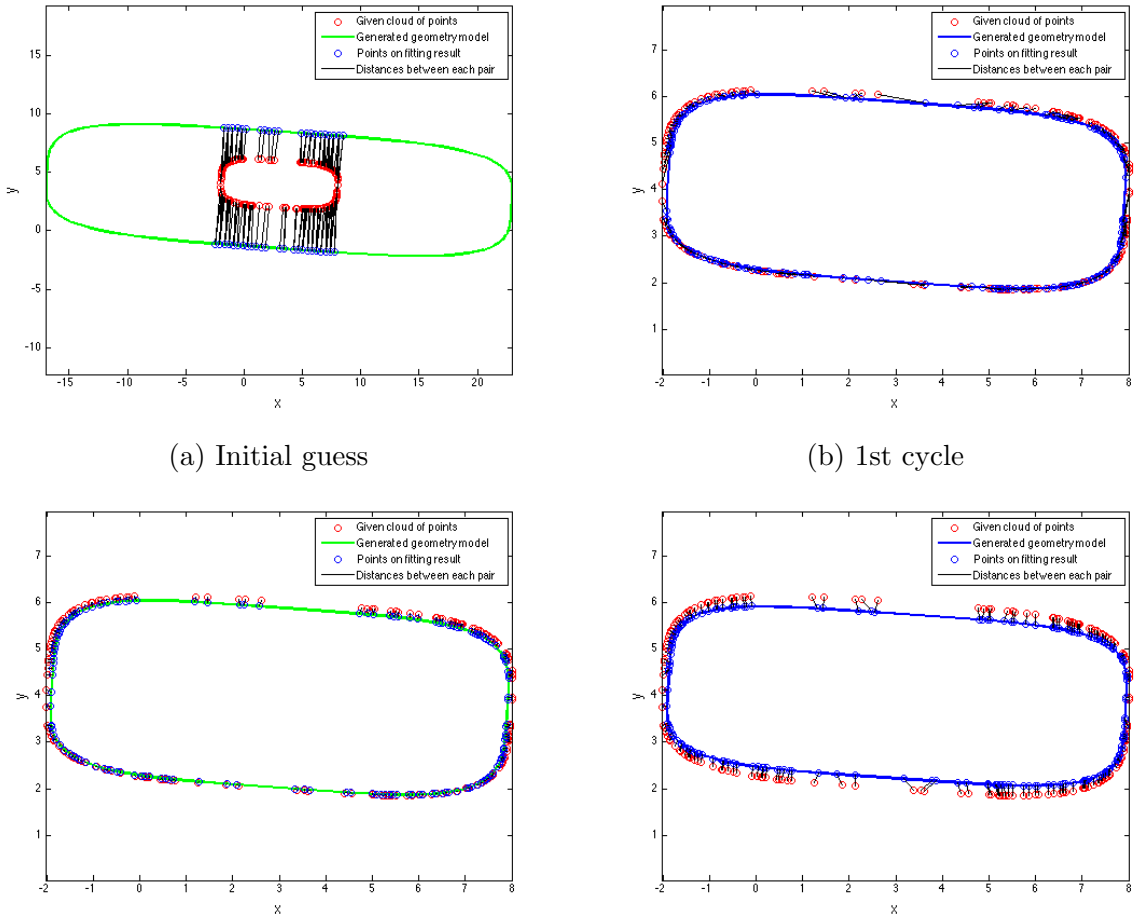
Figure 4.14: Fitting result analysis for too large initial guess problem

4.2.2 Add Penalty Term

For the super-ellipse fitting based on the large initial guess, an alternative penalty function can be defined. This penalty function is based on the idea that minimizes the distances to the geometry configuration and reduces the geometry shape simultaneously. The new

objective function after adding the penalty term is as Equation 4.7.

$$S = \sum_{i=1}^{m} \left(x_{p(i)} - f_x(\vec{d}, u_i)\right)^2 + \left(y_{p(i)} - f_y(\vec{d}, u_i)\right)^2 + a \times b \tag{4.7}$$



(a) Initial guess

(b) 1st cycle

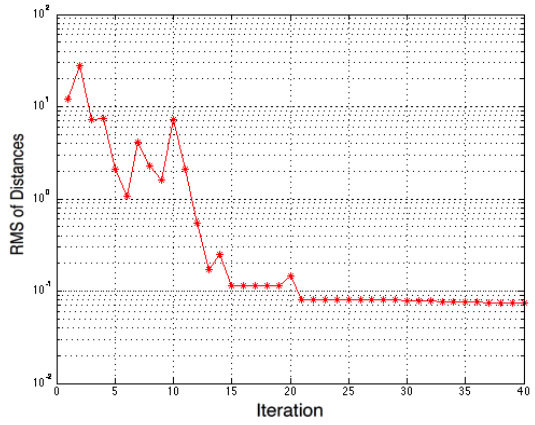
(c) Reinitialization result

(d) Final result

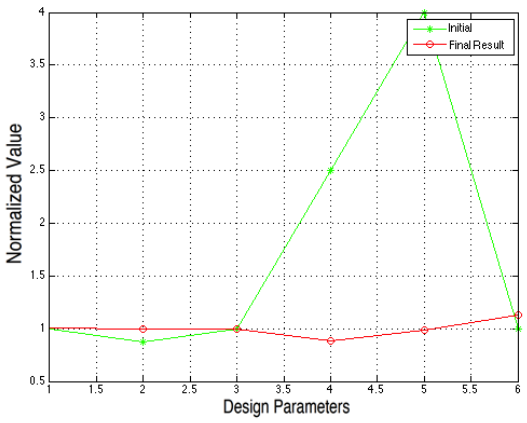
Figure 4.15: Results of super-ellipse generation after adding penalty term into objective function

As shown in the Figure 4.15, after 2 cycles, the resulting configuration is much closer to the cloud of points comparing with the result from ILM algorithm only. However, the result is smaller then the shape of points cloud. The reason is that the optimizer

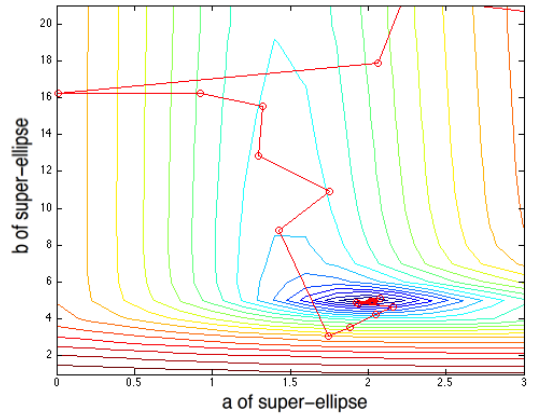
minimized not only the distances between the points in cloud and configuration, but also the value of $a \times b$. As in Figure 4.16d, although the slop of the plot is increased after adding the penalty term, the minimum value of the objective function is moved to the left of the original spot.



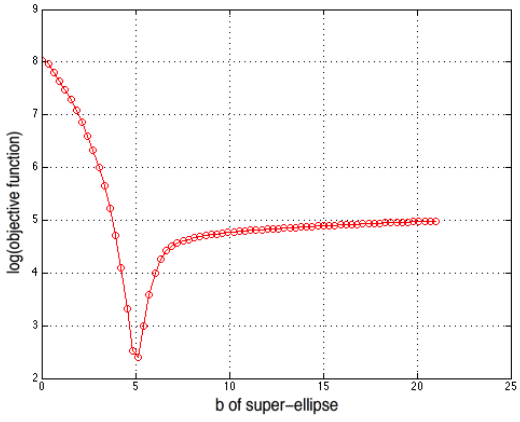
(a) RMS of distances



(b) Normalized design parameters



(c) Trajectory of design parameters a and b



(d) Objective function changed with b

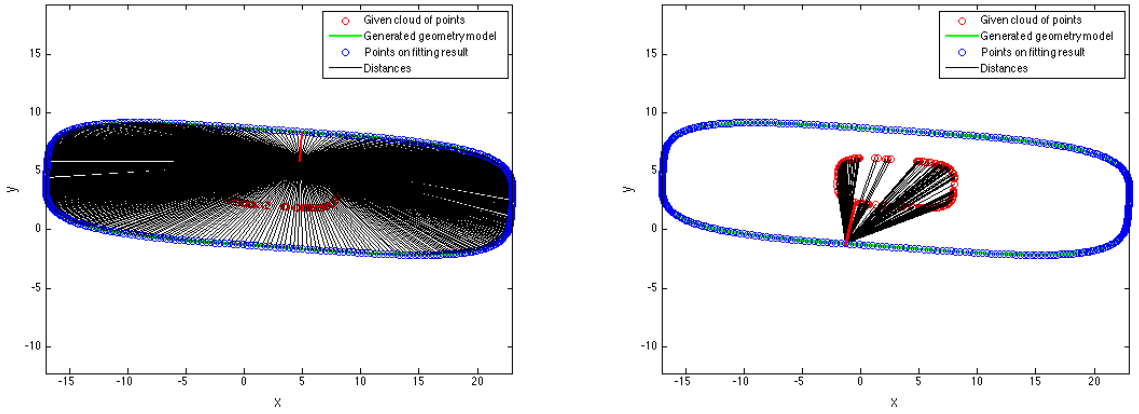
Figure 4.16: Fitting result analysis after adding penalty term into objective function

Although the result is smaller than the fitting target, this can be solved by removing the penalty term after several iterations during the optimization process. However, there is not a general rule that defines which design parameters should be added as a penalty

term into objective function. So, the more general penalty term need be defined for the optimization process.

4.2.3 New Objective Function

The basic idea of new objective function is that the points in the cloud should map to the full range of parametric coordinates. The specific method is, for each discrete point on the configuration, one searches the closest point from the cloud of points. This process can be added into the initialization technique as shown in Figure 4.17



(a) Distance from the original objective function (b) Distance from the penalty term

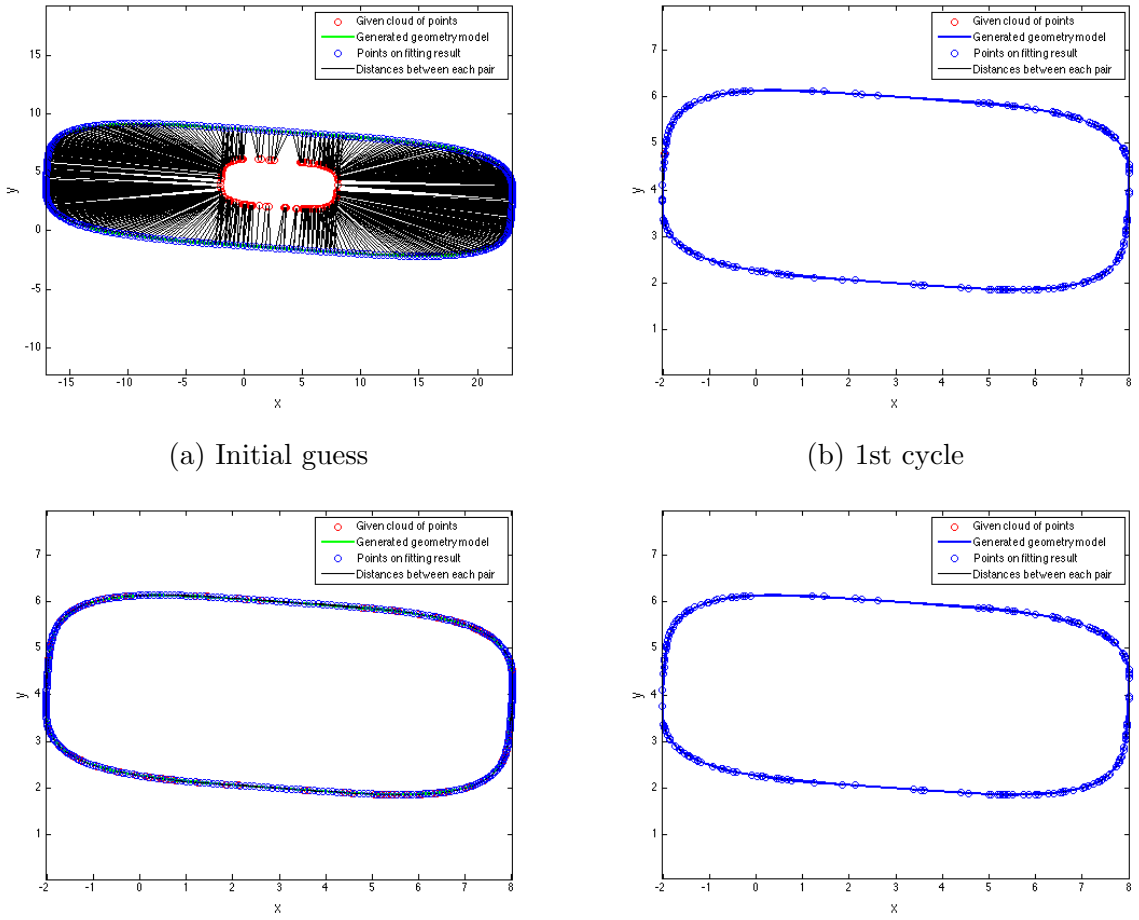
Figure 4.17: Components of new objective function

On one hand, Figure 4.17a shows the process that, for each point in the cloud, one searches the closest parametric coordinate on the configuration. And the sum of the square of these distances is the original objective function. On the other hand, Figure 4.17b shows the process that, for each parametric coordinate on the initial configuration, one searches the closest point form the cloud. And the sum of the square of these new distances is the penalty term of the objective function. After doing this, the points in

the cloud are mapped to the full range of the parametric coordinates. The new objective function is shown in Equation 4.8. In this format, the penalty term will be reduced to zero at the end of the optimization process (the parametric model result coincides with the points cloud). For improving the computing efficient, λ is a coefficient of penalty term that will be reduced to zero after 5 iterations. This can reduce the problem size and improve the computational time.

$$S = \sum_{i=1}^{m} \left(x_{p(i)} - f_x(\vec{d}, u_i)\right)^2 + \left(y_{p(i)} - f_y(\vec{d}, u_i)\right)^2 + \lambda \times \sum_{j=1}^{n} \left(x_{p(j)} - f_x(\vec{d}, u_j)\right)^2 + \left(y_{p(j)} - f_y(\vec{d}, u_j)\right)^2 \tag{4.8}$$

Figure 4.18 shows the fitting result after applying the new objective function. At this time, the accurate design parameters can be obtained only after 1 cycle. This not only overcame the bad initial guess problem, but also improved computational efficient.



(a) Initial guess

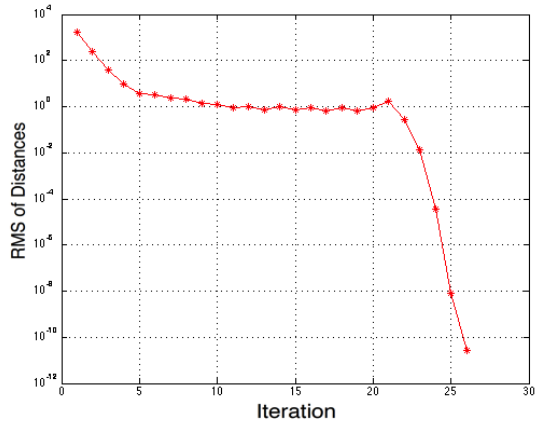
(b) 1st cycle

(c) Reinitialization result

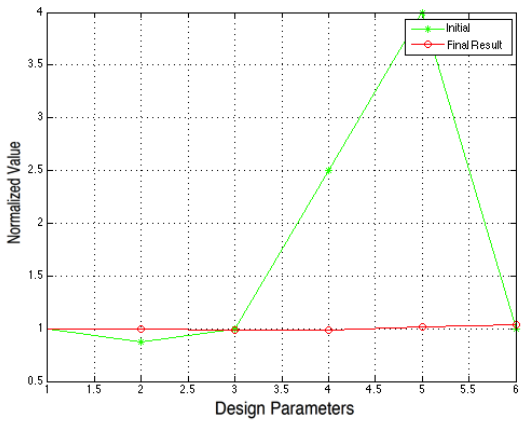
(d) Final result

Figure 4.18: Results of super-ellipse generation after using the new objective function

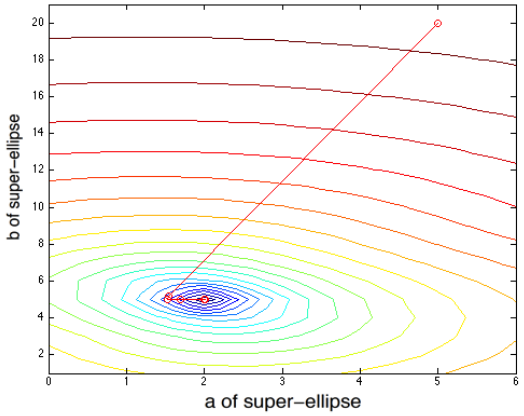
As shown in the Figure 4.19b, the normalized design parameters are equal to 1 after fitting process. In the Figure 4.19d, the slop of the curve changes to monotonically increased and is easier for optimizer getting the moving direction. The same conclusion can also be obtained from the Figure 4.19c. After applying the new objective function, more contour lines are added for providing enough direction information for optimizer.



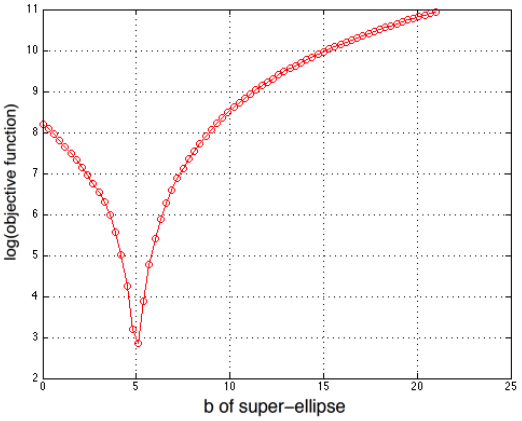
(a) RMS of distances



(b) Normalized design parameters



(c) Trajectory of design parameters a and b



(d) Objective function changed with b

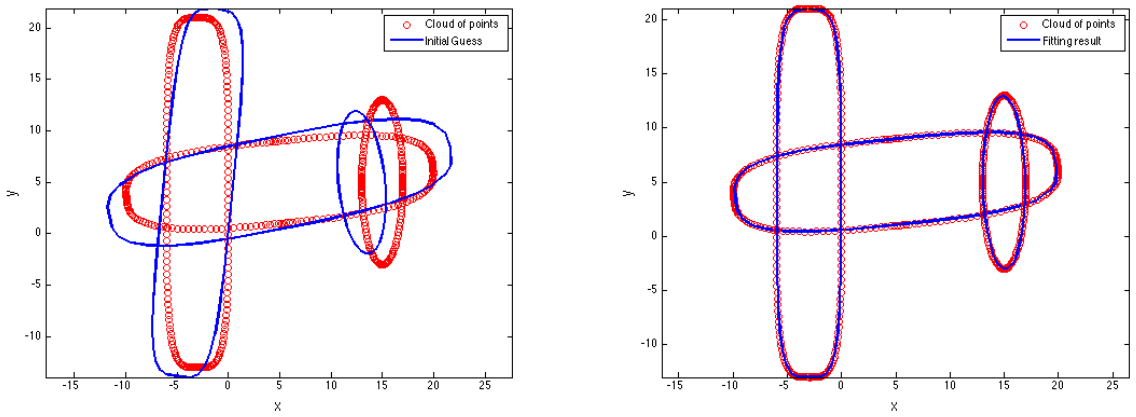
Figure 4.19: Fitting result analysis after using new objective function

Above content explained the solution for the larger initial guess problem. If the initial guess is smaller than the cloud of points, the new objective function is also a good solution. An additional problem for smaller initial guess is that, during the fitting iterations, it is possible for the parametric coordinates u to increase beyond the range over which u is defined; when this happens, the equations that generate the body shape cannot be evaluated. To ensure that this does not happen, it is important to define the parametric coordinates to be periodic.

4.3 (Re)Classification Technique

In the above, the fitting has been done for a single component. In this section, the generation of parametric models for multiple components based on a cloud of unclassified points is introduced. To ease the discussion, the technique will be explained in two dimensions with multiple super-ellipses in this section; it will then be applied to a three-dimensional case of a glider in section 3.5.2.

The basic multiple-component problem is demonstrated by three super-ellipses. As seen in Figure 4.20, the blue line in Figure 4.20a is the initial guess and the red points are the points in the cloud. The blue line in Figure 4.20b is three parametric models which is generated from the technique. The objective here is to find a method that starts from Figure 4.20a and ends at Figure 4.20b. This process is almost the same as the single component problem, but the multiple components problem needs one more technique to classify the different components and use the single component technique on each part after classification. Specifically, the classification need be finished at the same time as the objective function is minimized.



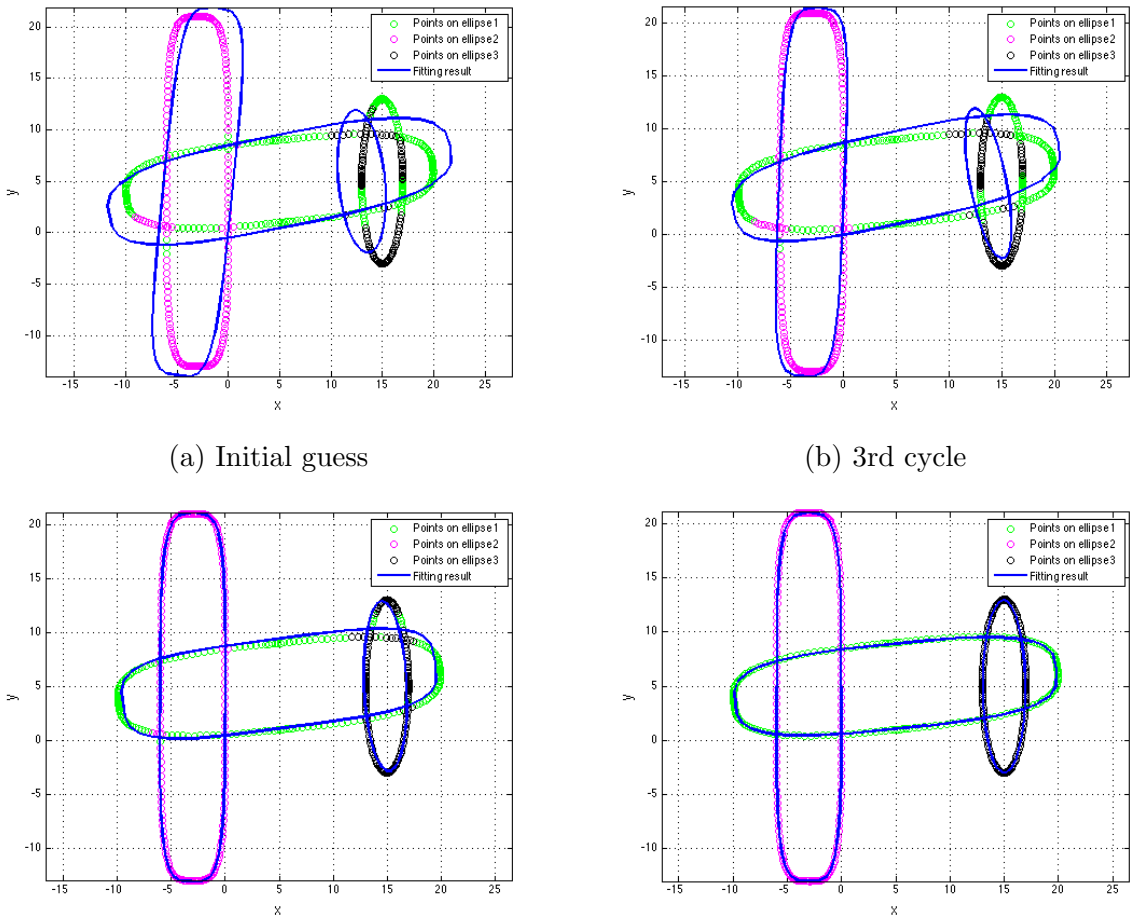
(a) Cloud of points and initial guess

(b) Fitting result

Figure 4.20: Sample classification problem for three super-ellipses

4.3.1 Basic Idea

Recall that for the single component problem, the original initialization technique found the u_i by finding the smallest distance from points in the cloud to the discrete points in the model. For multiple components, during the initialization process one also needs to record the identity of the component to which each point in the cloud belongs. Figure 4.21 shows the classification result based on different cycles. The pink points are classified to the first super-ellipse. The green points are classified to the second super-ellipse. The black points are classified to the third super-ellipse. At the beginning, there are many points that are mis-classified because of the poor initial guess. During the optimization process, the configuration matches the cloud of points better and the number of mis-classified points is reduced. At the end of the optimization process, the accurate parametric geometry model can be obtained and all points are classified to the correct related component.



(a) Initial guess

(b) 3rd cycle

(c) 7th cycle

(d) 10th cycles

Figure 4.21: Progression of fitting results for three super-ellipses (basic classification technique)

4.3.2 Improved Classification Technique

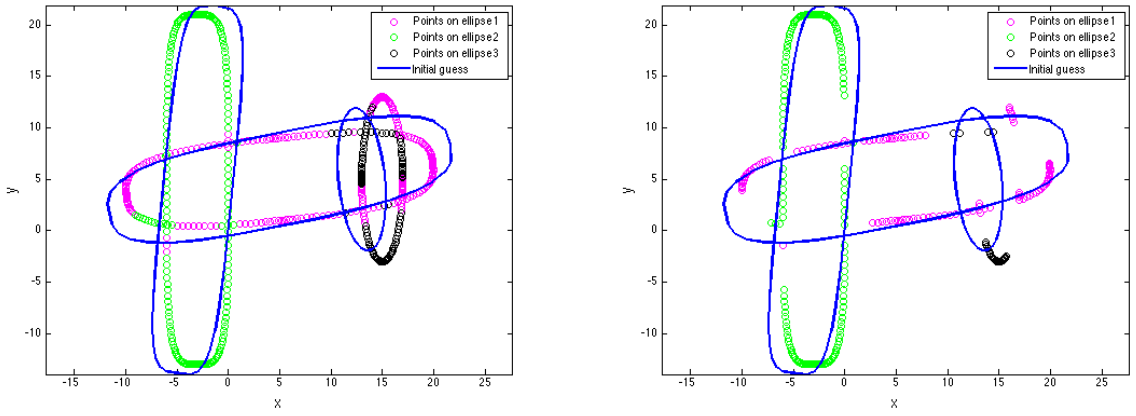
Classification correlates with the optimization process. If the initial classification is not good, the optimization will yield a bad result, which in turn will yield incorrect results in subsequent classifications. As can be seen in Figure 4.22a, there are many misclassified points near the intersections of the super-ellipses, which will adversely impact the optimization result. Therefore, a strategy has been adopted in which the points in the

cloud that are near the intersections (as evidenced by the fact that they are equally close to two or more components) are temporarily ignored; the tolerance to determine if they are "equally close" is rather loose in early stages and is gradually tightened so that, in the end, nearly all points are classified.

Here, we defined the coefficient of classification cof. Generally, cof is chosen the number between 1 to 2 at the beginning of the fitting process. The larger cof, the more junction points will be ignored during the classification process. When the $cof = 1$, there are no points ignored.

Note that classification is part of initialization, and that any time one needs to reinitialize, one also needs to reclassify.

This new classification scheme has been applied to the case of three intersecting super-ellipses, as shown in Figure 4.22b. The points in the junction area are ignored at the beginning of the optimization process.

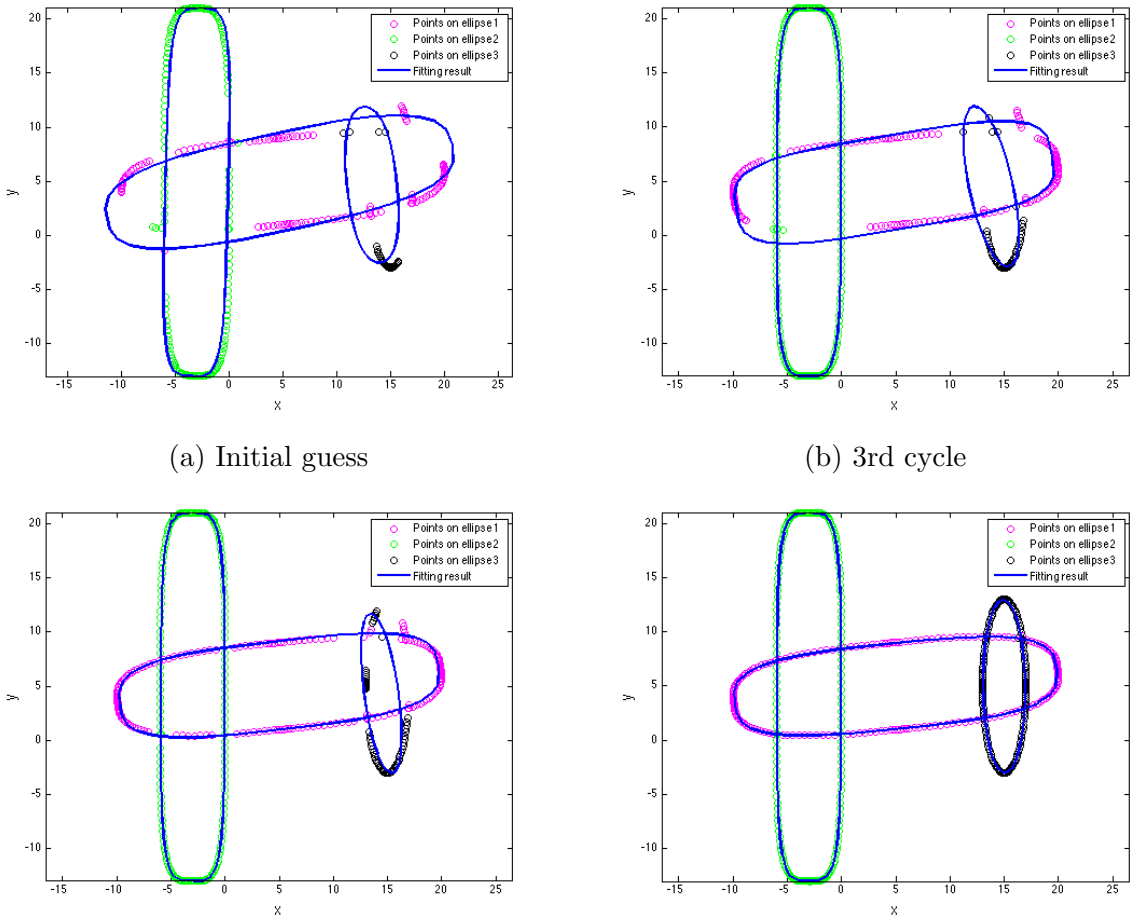


(a) Result of original classification

(b) Result of improved classification

Figure 4.22: Comparison of different classification results for three super-ellipses

Several cycles of the optimization are shown in Figure 4.23. The green points are the points classified to super-ellipse 1, the pink points are the points classified to super-ellipse 2, and the black points are the points classified to super-ellipse 3. The blue lines are the parametric super-ellipse model that is generated based on the cloud of points. During the optimization process, the configuration matches the cloud of points better and the ignored points in the junction area are counted back to the objective function. As shown in the Figure 4.23d, the accurate parametric geometry model can be obtained after 7 cycles. But for the original classification technique, there are 10 cycles for getting the correct design parameters. So, the improved classification technique improved the efficiency of the whole algorithms through avoiding the mis-classification problem.



(a) Initial guess

(b) 3rd cycle

(c) 5th cycle

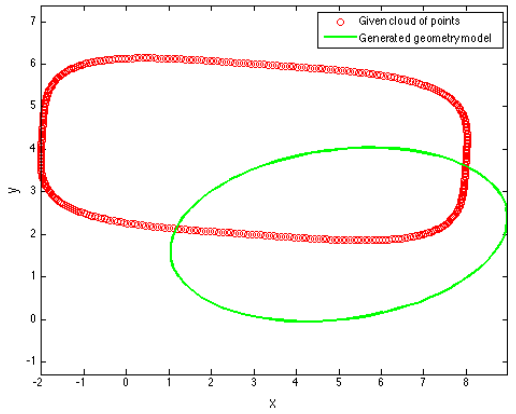
(d) 7th cycles

Figure 4.23: Progression of fitting results for three super-ellipses (improved classification technique)

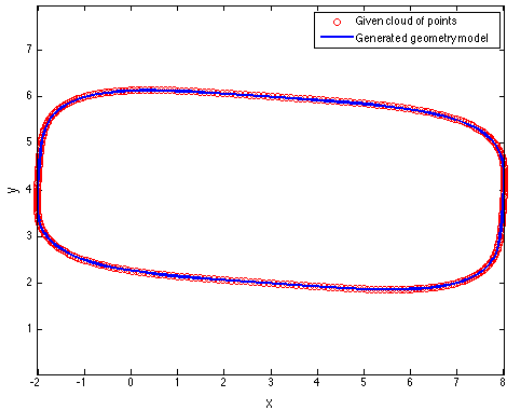
4.4 Algorithm Test

In this section, the data sensitivity of whole algorithm is evaluated. This is separated into two test parts. One is the test regarding the non-uniform cloud of points, the other one is regarding the noisy points in the cloud. Figure 4.24 shows the fitting result based on

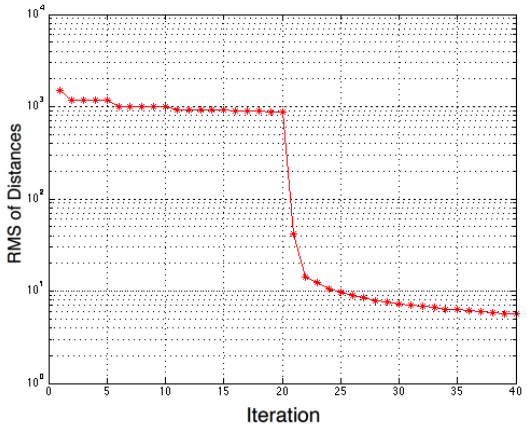
the general format of points cloud (uniform space and no noisy points). In this section, all the tests are upon 2 cycles and 20 maximum iterations per cycle.



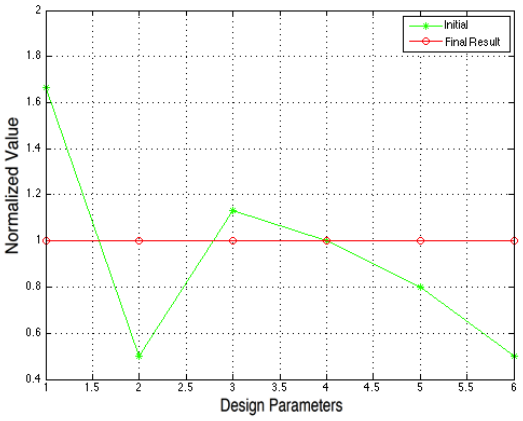
(a) Initial guess



(b) Fitting result



(c) RMS of distances



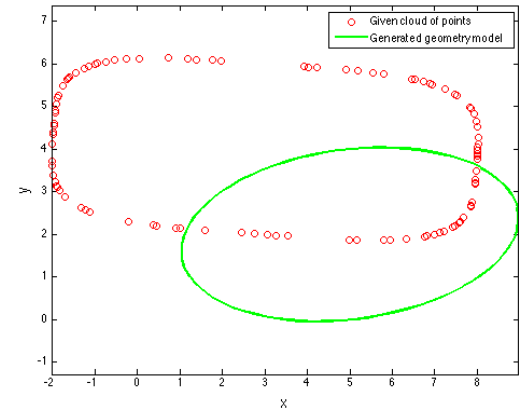
(d) Normalized design parameters

Figure 4.24: Super-ellipse fitting result based on the general format of points cloud

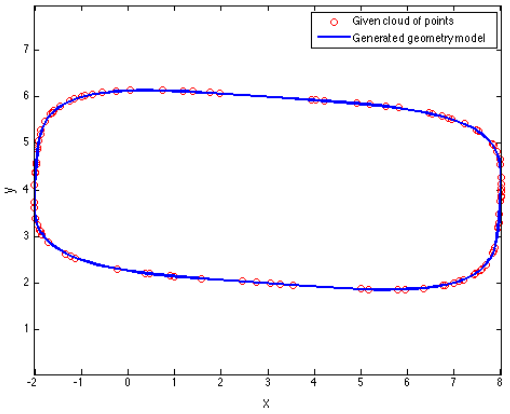
4.4.1 Non-uniform Cloud of Points

In this section, three non-uniform space points data are tested by the fitting algorithm. For the three cases, in Figure 4.25 4.26 4.27, points are randomly moved to non-uniform spots. This means that, in some areas, the points are overlapped, and in some areas,

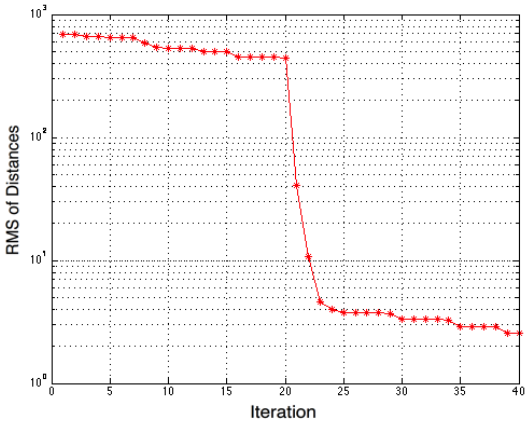
there are not points on it. As shown in the Figures, all of the fitting results are pretty close to the points in the cloud. And the normalized design parameters are equal to 1. Therefore, it can be concluded that the non-uniform cloud of points will not impact the accuracy of the algorithm.



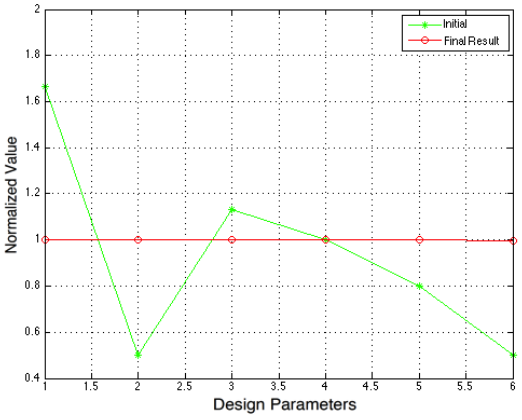
(a) Initial guess



(b) Fitting result

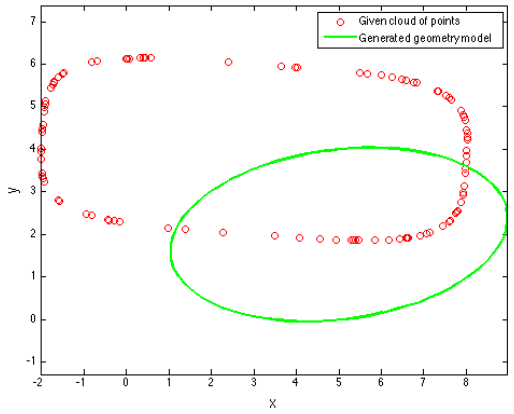


(c) RMS of distances

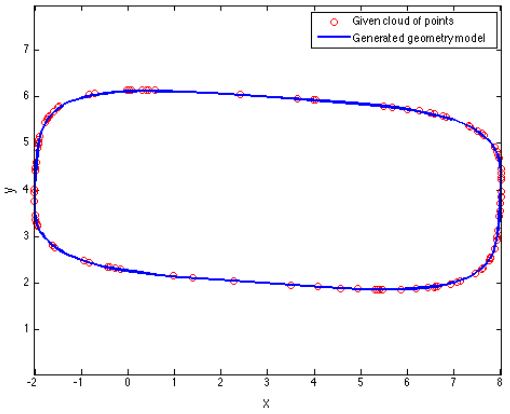


(d) Normalized design parameters

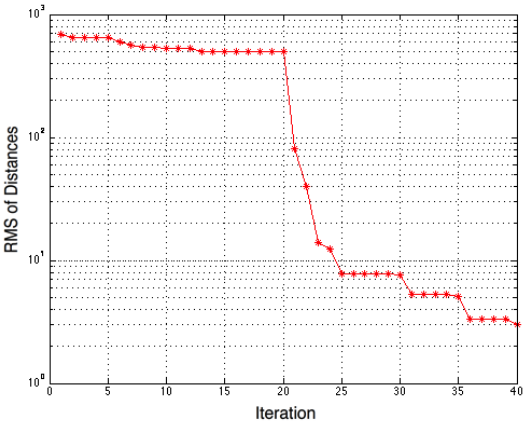
Figure 4.25: Super-ellipse fitting result based on the non-uniform space points data 1



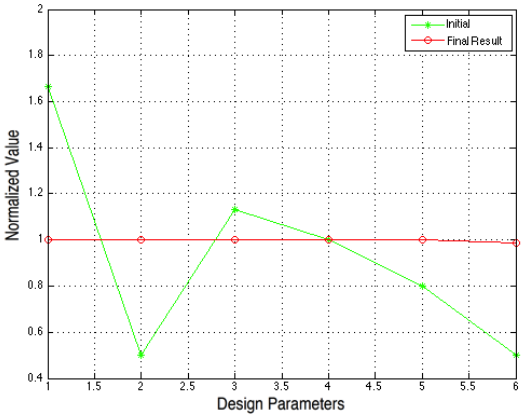
(a) Initial guess



(b) Fitting result

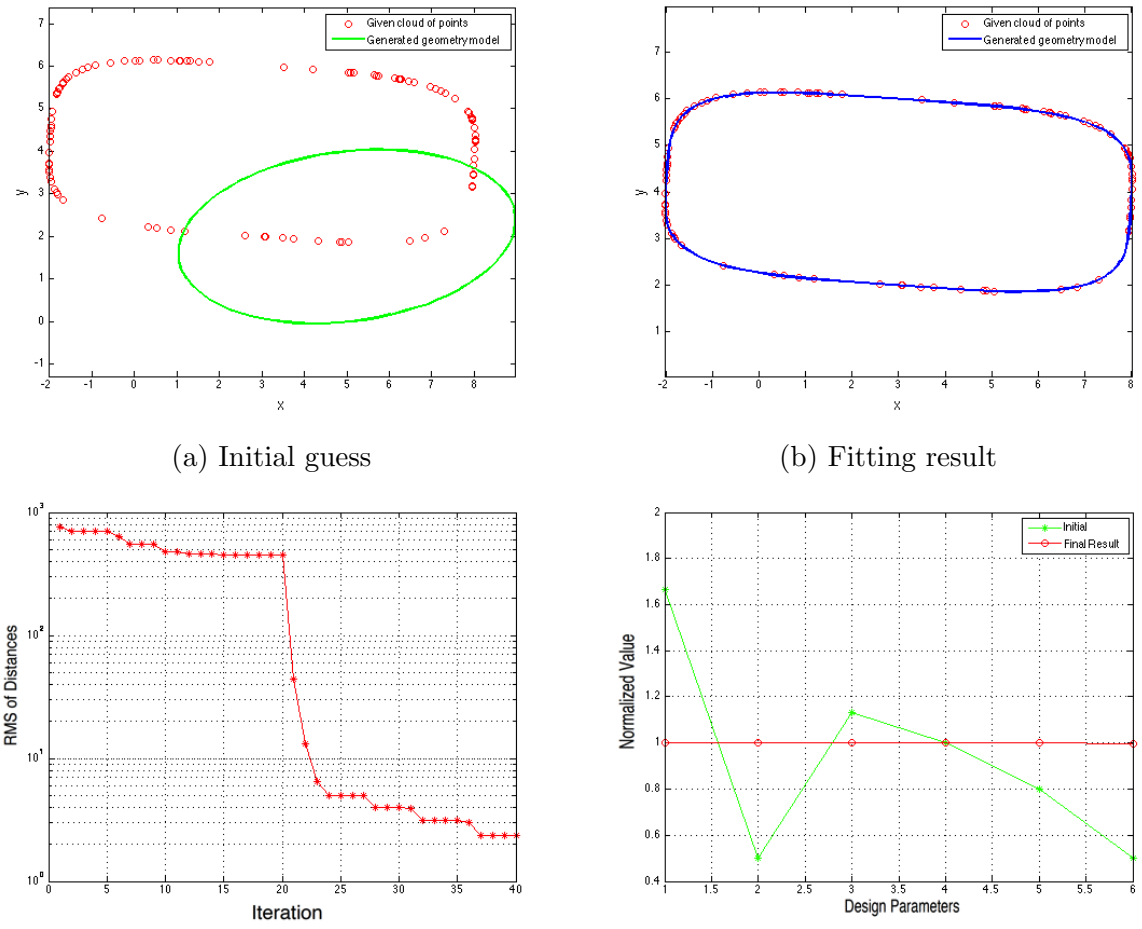


(c) RMS of distances



(d) Normalized design parameters

Figure 4.26: Super-ellipse fitting result based on the non-uniform space points data 2



(a) Initial guess

(b) Fitting result

(c) RMS of distances

(d) Normalized design parameters

Figure 4.27: Super-ellipse fitting result based on the non-uniform space points data 3

4.4.2 Noise Data Sensitivity

In this section, three noisy points data are tested by the algorithm. In each case, some errors are added into the original cloud of points. The method of adding errors is as below equation:

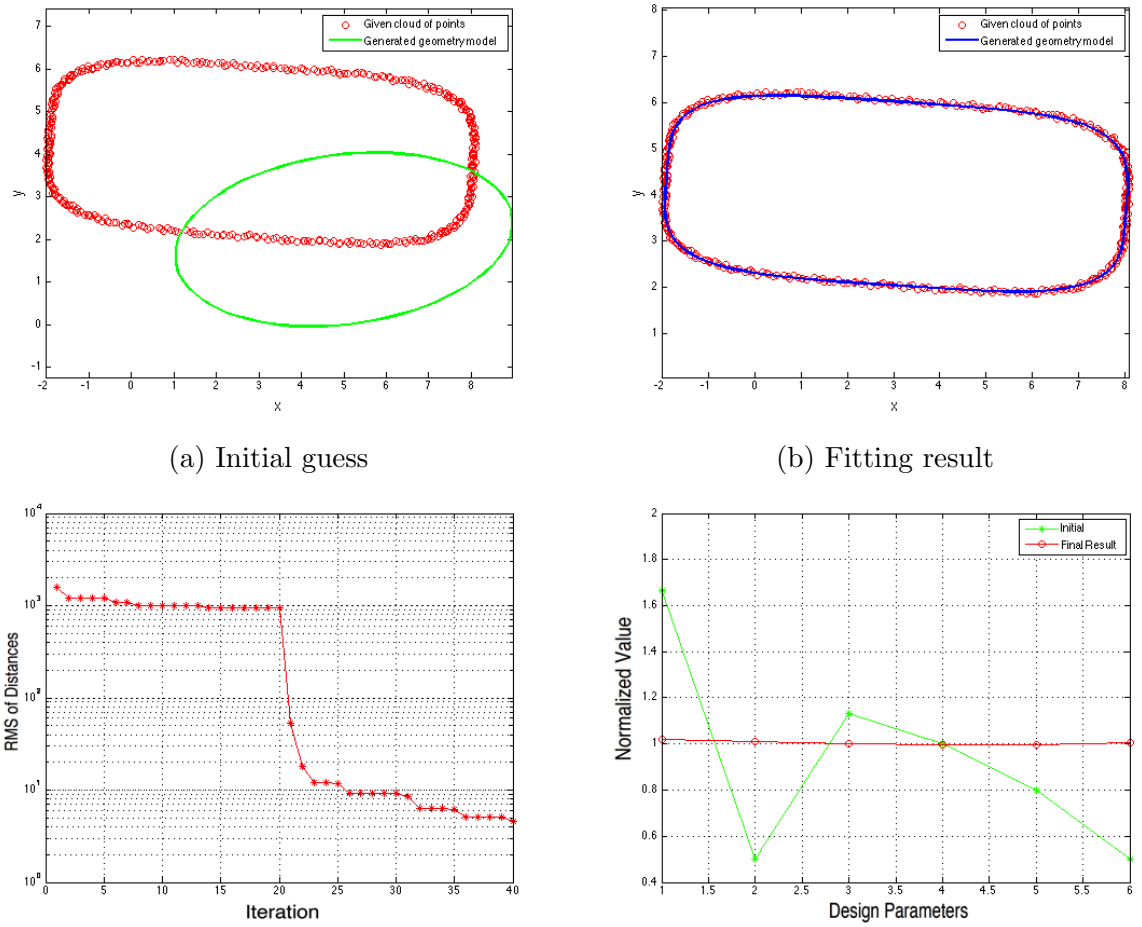
$$\begin{aligned} x_{pnew} &= err + x_p \\ y_{pnew} &= err + y_p \end{aligned} \tag{4.9}$$

where, the (x_{pnew}, y_{pnew}) are the coordinates of the point with noise. (x_p, y_p) are the original coordinates of the accurate points in the cloud. err is the random error added into the original coordinates. err is randomly distributed from $(0, 1)$. The mean of the err is 0.5. For the first case, in Figure 4.28, the adding err is $0.1 \times err$. Then we calculated the percentage of the error for the whole points in the cloud by the formula:

$$percent = 0.1 \times err/a \tag{4.10}$$

where, a is the width (short edge) of the super-ellipse.

So, the first case contains 1.6% errors as noisy points. The optimization process and the result are shown in Figure 4.28. The normalized design parameters are equal to 1 even if there are noisy points in the cloud.



(a) Initial guess

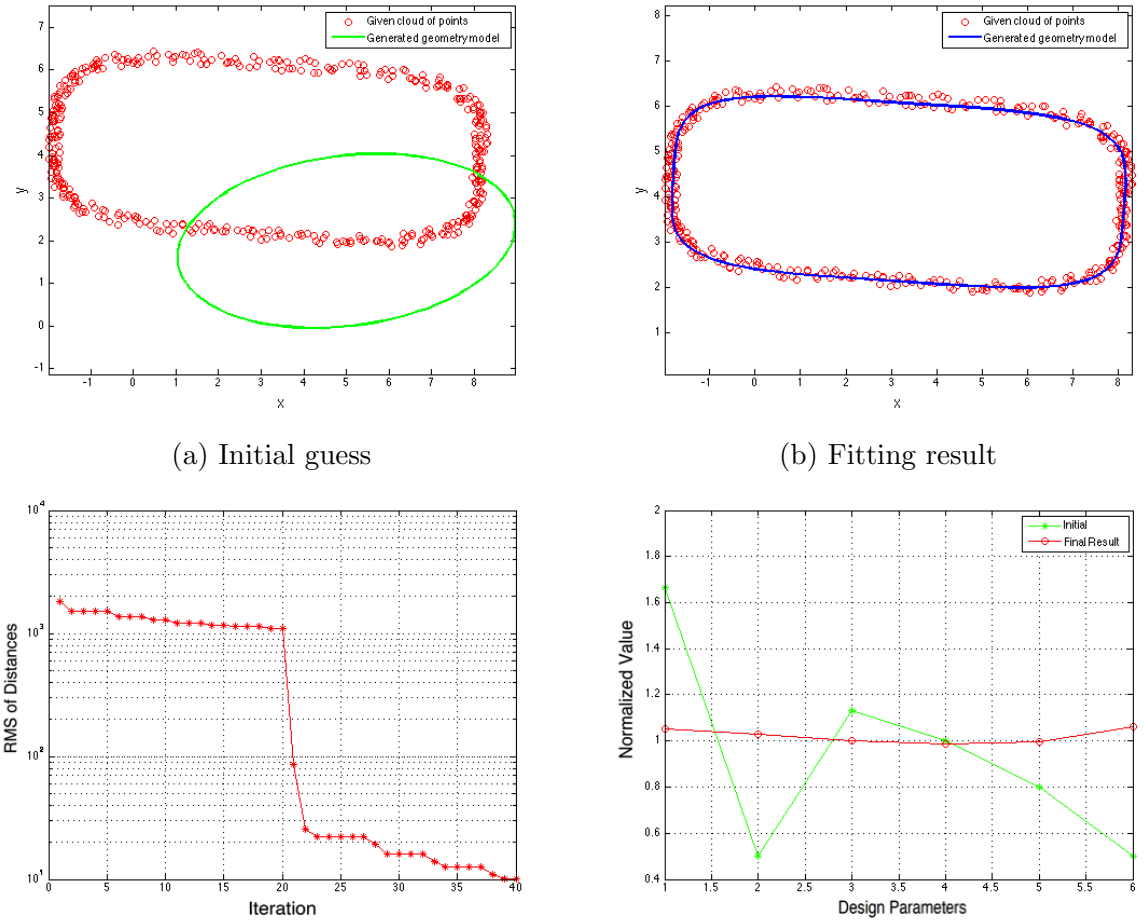
(b) Fitting result

(c) RMS of distances

(d) Normalized design parameters

Figure 4.28: Super-ellipse fitting result based on the noisy points data 1

For the second testing case, 5% errors are contained in the cloud of points. As shown in the Figure 4.29, the normalized result design parameters are moved away from 1 a little bit, but the results are still in the acceptable range.



(a) Initial guess

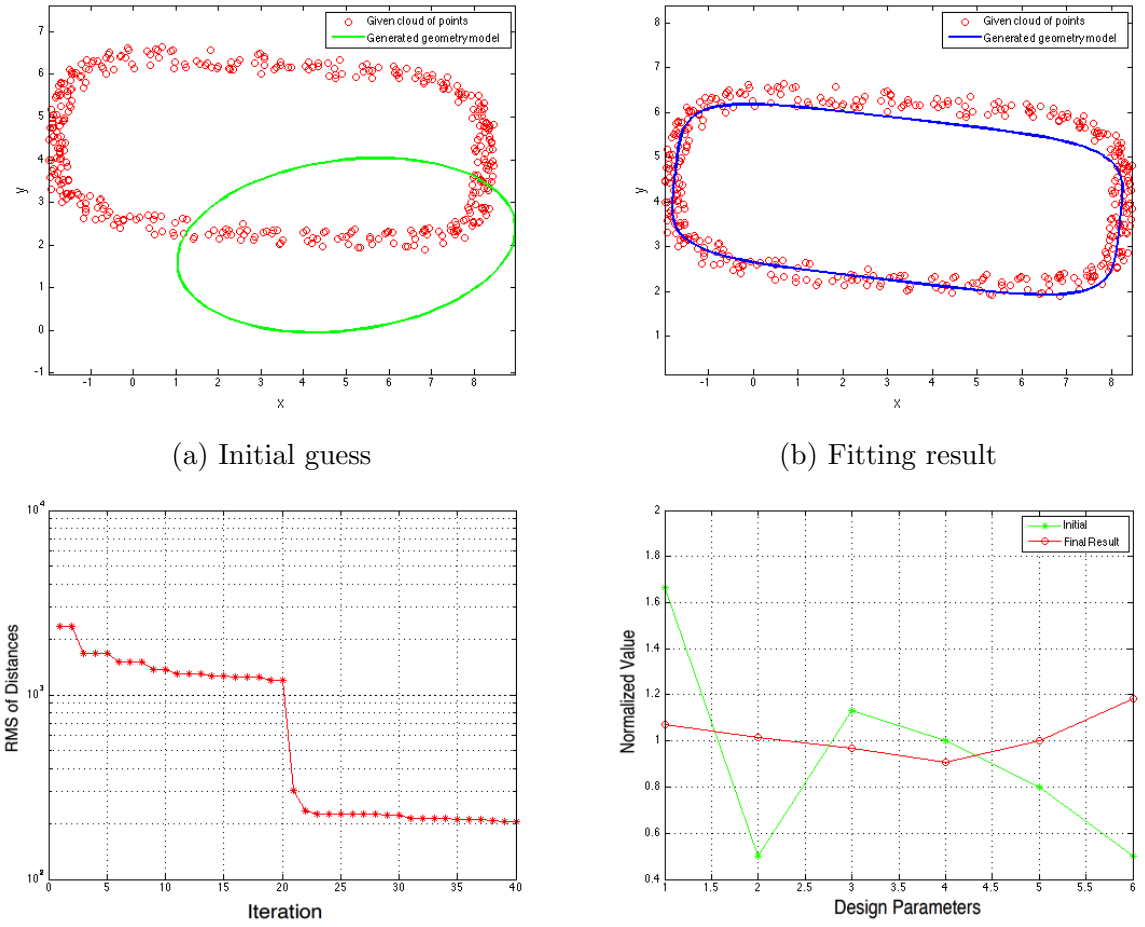
(b) Fitting result

(c) RMS of distances

(d) Normalized design parameters

Figure 4.29: Super-ellipse fitting result based on the noisy points data 2

For the third testing case, 8.3% errors are contained in the cloud of points. As shown in the Figure 4.30, the normalized result design parameters are moved further from 1. Although the rough parametric geometry model can be obtained from this case, the design parameters are not accurate enough.



(a) Initial guess

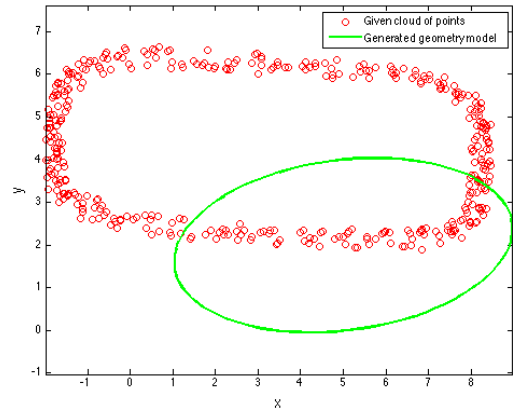
(b) Fitting result

(c) RMS of distances

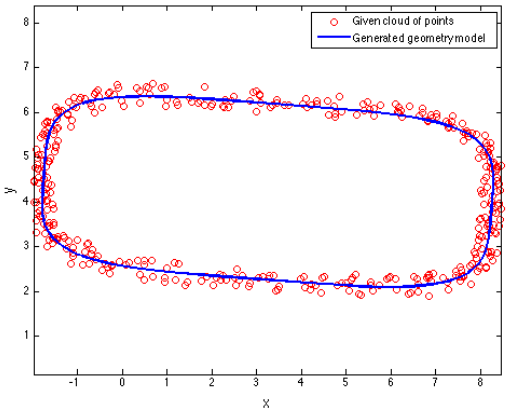
(d) Normalized design parameters

Figure 4.30: Super-ellipse fitting result based on the noisy points data 3

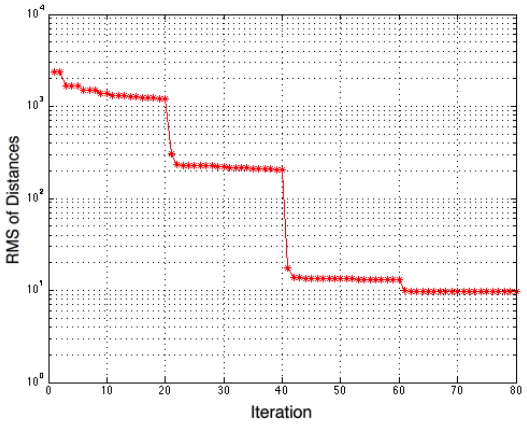
However, for the further testing, two more extra cycles (total 4 cycles) are run on this case. The fitting results are shown in Figure 4.31. As shown in the figure, the normalized design parameters are more closer to 1 comparing with only run two cycles. And the RMS is still reduced in the extra 2 cycles.



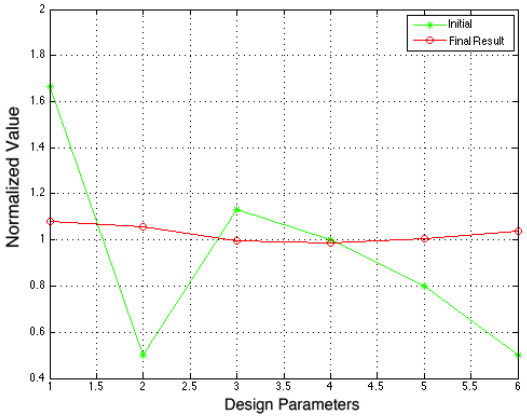
(a) Initial guess



(b) Fitting result (extra 2 cycles)



(c) RMS of distances



(d) Normalized design parameters

Figure 4.31: Super-ellipse fitting result based on the noisy points data 3 (run extra 2 cycles)

Therefore, it can be concluded that it needs more running time (cycles) when the cloud of points contains more errors or noisy points. In order to getting the accurate parametric model via this algorithm, the efficiency (running time) should be used as trade off term.

4.5 Test Examples in 3D

In this section, the whole algorithm will be tested in 3D configuration generation problem. Comparing with the 2D problem, 3D problem have 2 parametric coordinates (u_i, v_i) for each point in the space. Here, we recall the Jacobian matrix defined in Chapter 3, section 2.3.1. The Jacobian matrix for 3D (super-ellipsoid) problem can be written as:

$$J = -\left[\begin{array}{ccc|cccccccc}
\frac{\partial f_{x(1)}}{\partial d_1} & \cdots & \frac{\partial f_{x(1)}}{\partial d_n} & \frac{\partial f_{x(1)}}{\partial u_1} & 0 & \cdots & 0 & \frac{\partial f_{x(1)}}{\partial v_1} & 0 & \cdots & 0 \\
\frac{\partial f_{x(2)}}{\partial d_1} & \cdots & \frac{\partial f_{x(2)}}{\partial d_n} & 0 & \frac{\partial f_{x(2)}}{\partial u_2} & \cdots & 0 & 0 & \frac{\partial f_{x(2)}}{\partial v_2} & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_{x(m)}}{\partial d_1} & \cdots & \frac{\partial f_{x(m)}}{\partial d_n} & 0 & \cdots & \cdots & \frac{\partial f_{x(m)}}{\partial u_m} & 0 & \cdots & \cdots & \frac{\partial f_{x(m)}}{\partial v_m} \\
\frac{\partial f_{y(1)}}{\partial d_1} & \cdots & \frac{\partial f_{y(1)}}{\partial d_n} & \frac{\partial f_{y(1)}}{\partial u_1} & 0 & \cdots & 0 & \frac{\partial f_{y(1)}}{\partial v_1} & 0 & \cdots & 0 \\
\frac{\partial f_{y(2)}}{\partial d_1} & \cdots & \frac{\partial f_{y(2)}}{\partial d_n} & 0 & \frac{\partial f_{y(2)}}{\partial u_2} & \cdots & 0 & 0 & \frac{\partial f_{y(2)}}{\partial v_2} & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_{y(m)}}{\partial d_1} & \cdots & \frac{\partial f_{y(m)}}{\partial d_n} & 0 & \cdots & \cdots & \frac{\partial f_{y(m)}}{\partial u_m} & 0 & \cdots & \cdots & \frac{\partial f_{y(m)}}{\partial v_m} \\
\frac{\partial f_{z(1)}}{\partial d_1} & \cdots & \frac{\partial f_{z(1)}}{\partial d_n} & \frac{\partial f_{y(1)}}{\partial u_1} & 0 & \cdots & 0 & \frac{\partial f_{z(1)}}{\partial v_1} & 0 & \cdots & 0 \\
\frac{\partial f_{z(2)}}{\partial d_1} & \cdots & \frac{\partial f_{z(2)}}{\partial d_n} & 0 & \frac{\partial f_{z(2)}}{\partial u_2} & \cdots & 0 & 0 & \frac{\partial f_{z(2)}}{\partial v_2} & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_{z(m)}}{\partial d_1} & \cdots & \frac{\partial f_{z(m)}}{\partial d_n} & 0 & \cdots & \cdots & \frac{\partial f_{z(m)}}{\partial u_m} & 0 & \cdots & \cdots & \frac{\partial f_{z(m)}}{\partial v_m}
\end{array}\right] \tag{4.11}$$

After using spare technique, the Equation 4.12 can be stored as:

$$J = -\left[\begin{array}{ccc|cc} \frac{\partial f_{x(1)}}{\partial d_1} & \cdots & \frac{\partial f_{x(1)}}{\partial d_n} & \frac{\partial f_{x(1)}}{\partial u_1} & \frac{\partial f_{x(1)}}{\partial v_1} \\ \frac{\partial f_{x(2)}}{\partial d_1} & \cdots & \frac{\partial f_{x(2)}}{\partial d_n} & \frac{\partial f_{x(2)}}{\partial u_2} & \frac{\partial f_{x(2)}}{\partial v_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_{x(m)}}{\partial d_1} & \cdots & \frac{\partial f_{x(m)}}{\partial d_n} & \frac{\partial f_{x(m)}}{\partial u_m} & \frac{\partial f_{x(m)}}{\partial v_m} \\ \frac{\partial f_{y(1)}}{\partial d_1} & \cdots & \frac{\partial f_{y(1)}}{\partial d_n} & \frac{\partial f_{y(1)}}{\partial u_1} & \frac{\partial f_{y(1)}}{\partial v_1} \\ \frac{\partial f_{y(2)}}{\partial d_1} & \cdots & \frac{\partial f_{y(2)}}{\partial d_n} & \frac{\partial f_{y(2)}}{\partial u_2} & \frac{\partial f_{y(2)}}{\partial v_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_{y(m)}}{\partial d_1} & \cdots & \frac{\partial f_{y(m)}}{\partial d_n} & \frac{\partial f_{y(m)}}{\partial u_m} & \frac{\partial f_{y(m)}}{\partial v_m} \\ \frac{\partial f_{z(1)}}{\partial d_1} & \cdots & \frac{\partial f_{z(1)}}{\partial d_n} & \frac{\partial f_{y(1)}}{\partial u_1} & \frac{\partial f_{z(1)}}{\partial v_1} \\ \frac{\partial f_{z(2)}}{\partial d_1} & \cdots & \frac{\partial f_{z(2)}}{\partial d_n} & \frac{\partial f_{z(2)}}{\partial u_2} & \frac{\partial f_{z(2)}}{\partial v_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_{z(m)}}{\partial d_1} & \cdots & \frac{\partial f_{z(m)}}{\partial d_n} & \frac{\partial f_{z(m)}}{\partial u_m} & \frac{\partial f_{z(m)}}{\partial v_m} \end{array}\right] \tag{4.12}$$

In this section, the fuselage and wing are tested as a single component configuration, because they are the basic components of aircraft.

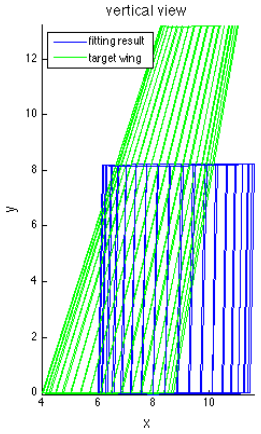
4.5.1 Single Component Configuration

Figure 4.32 shows the fitting result of 3D wing on different cycles based on 1000 points. The green lines are the fitting target (cloud of points), which is the original wing model that was used to generate the cloud of points, and the blue lines show the optimization results. The convergence histories of these calculations are shown in Figure 4.33. In part (a) the RMS of the distances between the points in the cloud and the surface are shown as a function of iteration number; the abrupt rises in the RMS values occur during the reinitialization process (because the initial values of (u_i, v_i) are restricted to being one of the points in the discrete representation of the model).

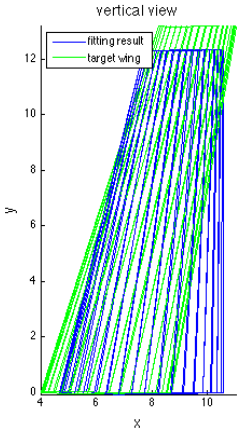
In part (b), the various design parameters, each normalized by its value that was used when the cloud of points were generated, are shown. Most of the optimized design parameters match their targets very well; the few that do not fit well (wing area, twist angle, and maximum thickness) turn out to have very little influence in this case and so their discrepancies are not significant.

Table 4.1: Design Parameters of the Wing

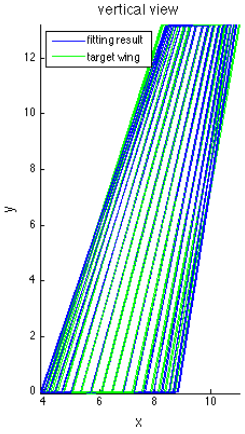
X Loc	Z Loc	Thick	Camber	Area	Aspect	Taper	Sweep	Twist	Dihedral
4	2	0.12	0.04	8	5	0.4	15	5	4



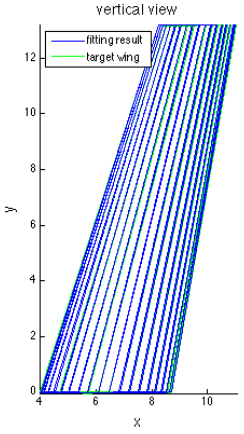
(a) Initial value of wing



(b) Fitting result after 1 cycle

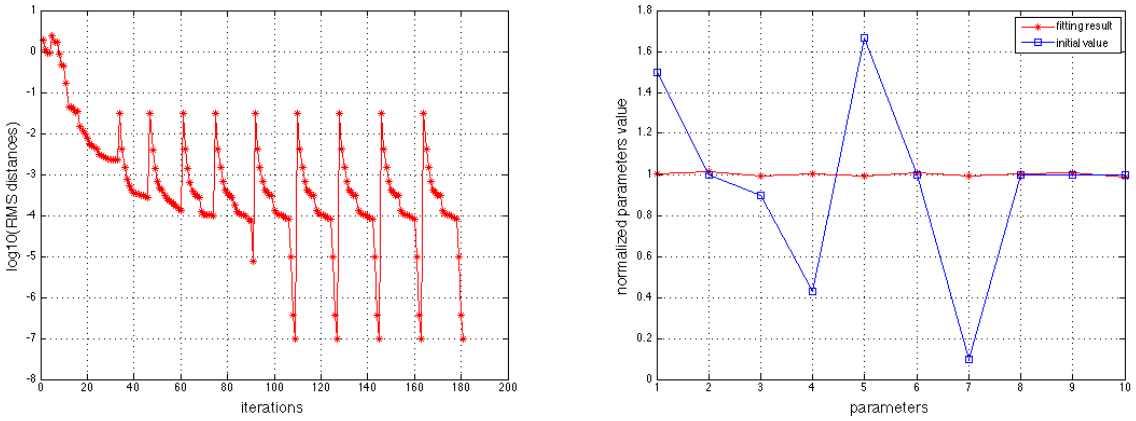


(c) Fitting result after 2 cycles



(d) Fitting result after 3 cycles

Figure 4.32: Progression of fitting results for 3D wing



(a) RMS of distances (b) Normalized parameters

Figure 4.33: RMS distances and normalized parameters for 3D wing

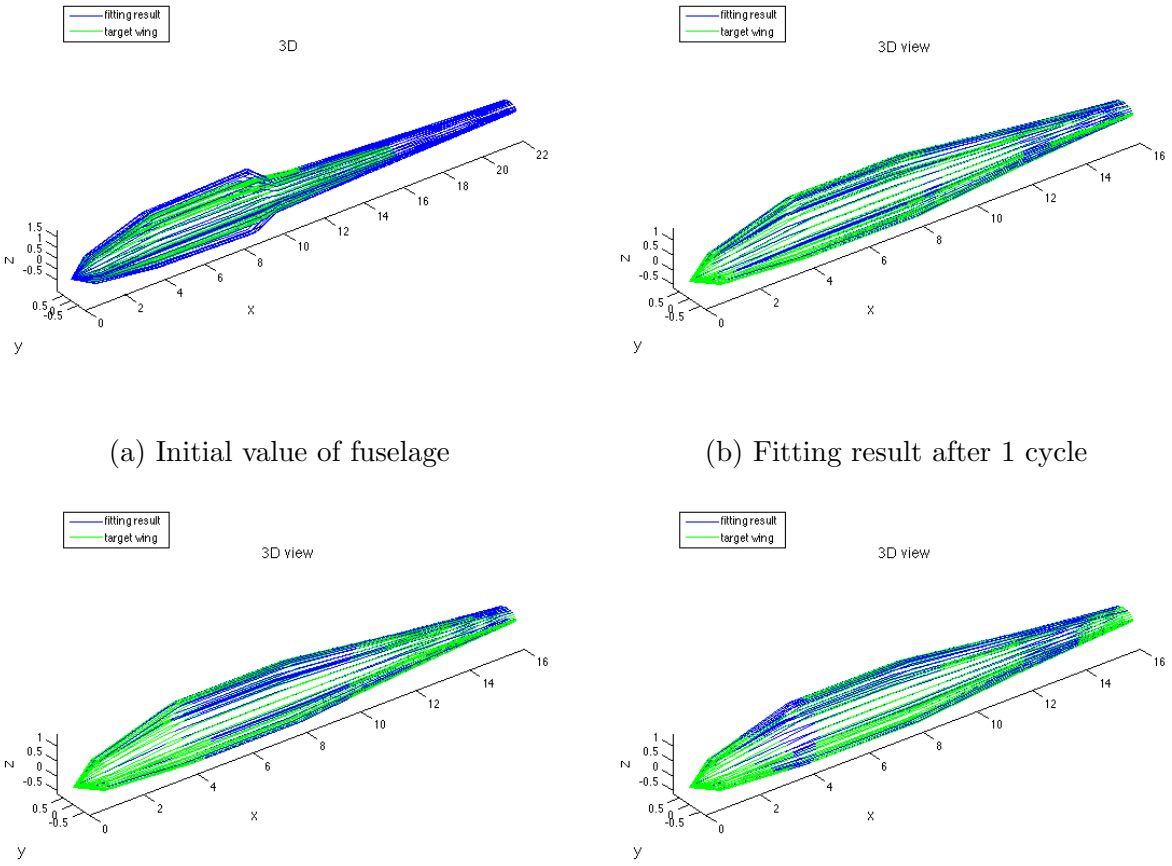
The second single-component configuration in three dimensions is a parametric fuselage, which is defined as the ruled surface between several super-elliptical cross-sections. Since there are 6 cross-sections, and each has 4 design parameters, this case contains a total of 24 design parameters, which are shown in Table 4.2. These are in addition to the $2m$ optimization variables (u_i, v_i) that are associated with the m points in the cloud.

Table 4.2: Design Parameters of the Fuselage

	Section1	Section2	Section3	Section4	Section5	Section6
X Location	0	1	4	8	12	16
Z Location	0	0.1	0.4	0.4	0.3	0.2
Width	0	1	1.6	1.6	1	0.8
Height	0	1	2	2	1.2	0.4

The iteration history, final fitting result, and normalized optimized parameters are shown in Figures 4.34-4.36. As can be seen from the Figure 4.34, the correct fitting result can

be obtained after one cycle. For Figure 4.36, the discontinuities result from target values that are zero, for which a normalized result is not defined.



(a) Initial value of fuselage

(b) Fitting result after 1 cycle

(c) Fitting result after 2 cycles

(d) Fitting result after 3 cycles

Figure 4.34: Progression of fitting results for 3D fuselage

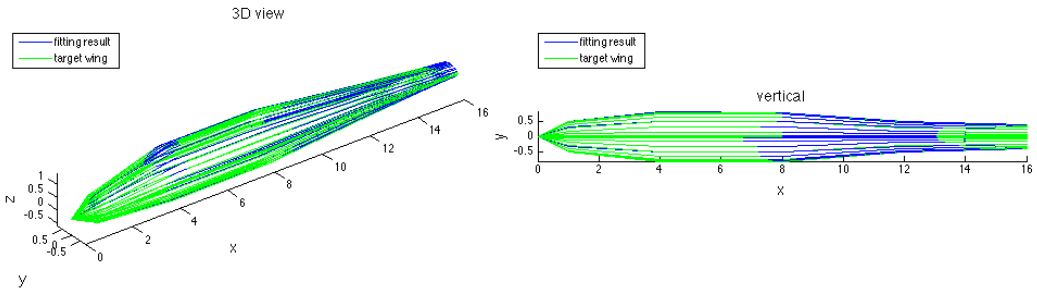
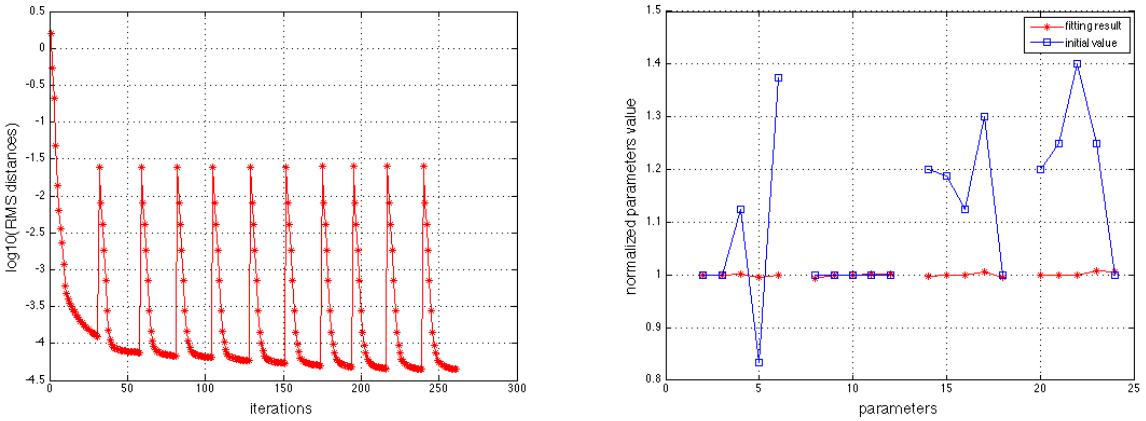


Figure 4.35: Final fitting result of 3D fuselage after using periodic (u, v)



(a) RMS of distances

(b) Normalized parameters

Figure 4.36: RMS distances and normalized parameters for 3D fuselage

4.5.2 Multiple Components Configuration

Figures 4.37 to 4.40 show the results of fitting the glider to the cloud of points with the classification technique. Because the initial guess is used in the first classification, care must be taken to create an initial guess that is somewhat close to the final configuration. (For example, an aircraft with a canard configuration should start with design parameters for a canard and not parameters for a conventional wing/tail configuration.) The initial

guess and generated model for different cycles are shown in Figure 4.37. The green lines are the target model and blue lines are the generated glider model. The correct fitting result is obtained after 8 cycles. In this case, the coefficient of classification technique cof is set as 2. After 6 cycles, this coefficient cof is reduced to 1 which means no points are ignored. The original design parameters of glider are shown in Table 5.5 - 5.8

Table 4.3: Design Parameters of the the Fuselage in Glider

	Section1	Section2	Section3	Section4	Section5	Section6
X Location	0	1	4	8	12	16
Z Location	0	0.1	0.4	0.4	0.3	0.2
Width	0	1	1.6	1.6	1	0.8
Height	0	1	2	2	1.2	0.4

Table 4.4: Design Parameters of the Wing in Glider

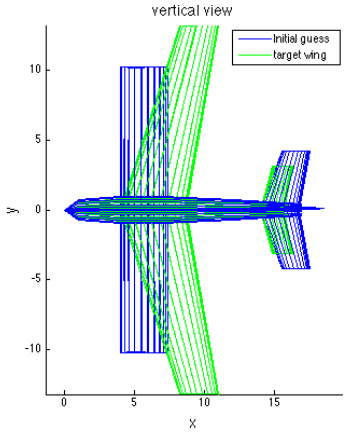
X Location	Z Location	Thick	Camber	Area	Aspect	Taper	Sweep	Twist	Dihedral
4	0.2	0.12	0.04	100	7	0.6	10	-5	5

Table 4.5: Design Parameters of the Horizontal Tail in Glider

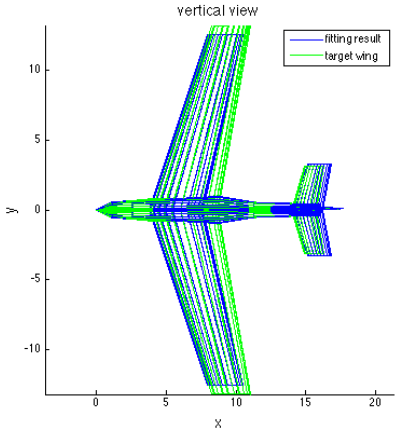
X Location	Z Location	Thick	Camber	Area	Aspect	Taper	Sweep	Twist	Dihedral
14	0.2	0.1	0	10	4	0.8	10	0	0

Table 4.6: Design Parameters of the Vertical Tail in Glider

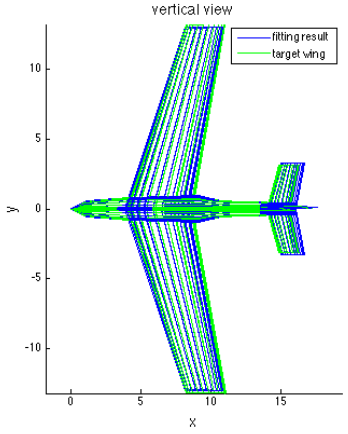
X Location	Z Location	Thick	Camber	Area	Aspect	Taper	Sweep
13.5	0.1	0.1	0	10	3	0.5	30



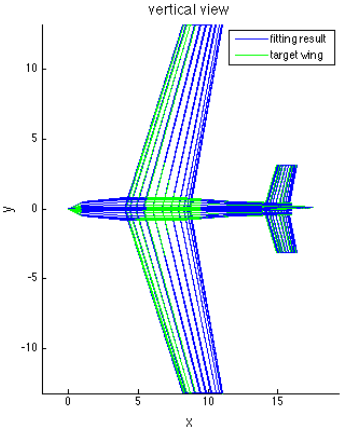
(a) Initial value



(b) Fitting result after 1st cycle



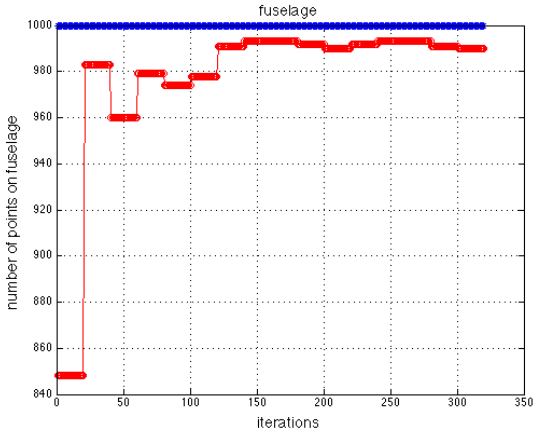
(c) Fitting result after 2nd cycle



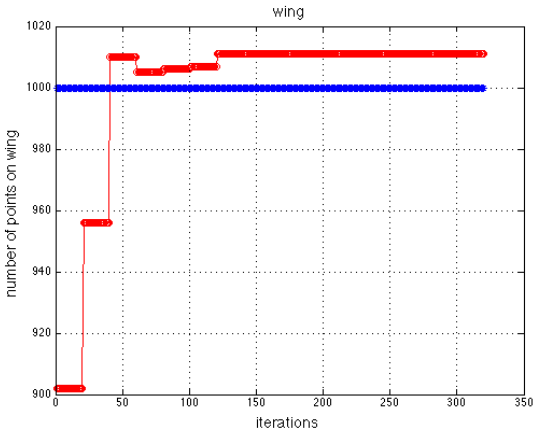
(d) Fitting result after 8th cycle

Figure 4.37: Progression of fitting results for 3D glider

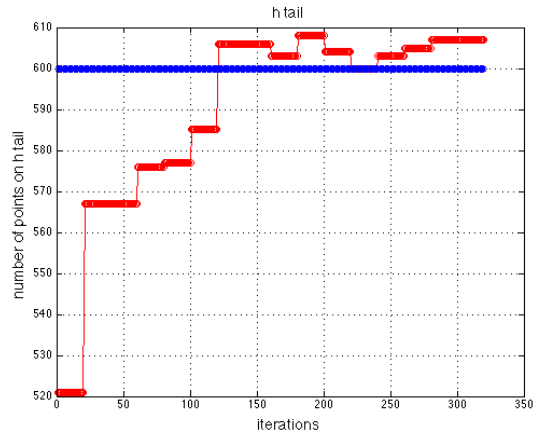
Figure 4.38 shows the number of points associated with each component after classification cycle. Because the points in the vicinity of the intersections are be ignored at the beginning of optimization, the number of points associated with each component is much smaller than it should be. As the optimization progresses, and the shape of the model becomes closer to the correct result, the number of ignored points is automatically reduced. The final number of points associated with each component ends up being within 1% of the correct number.



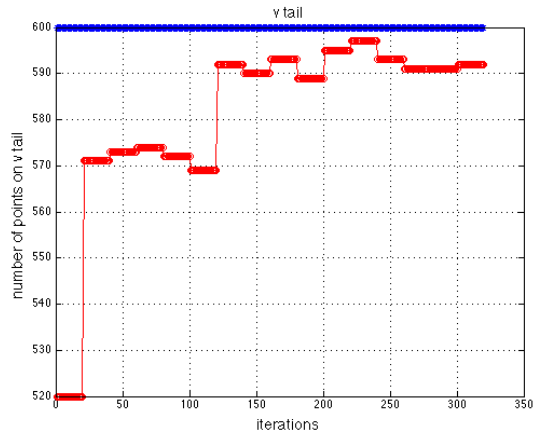
(a) Number of points on fuselage



(b) Number of points on wing



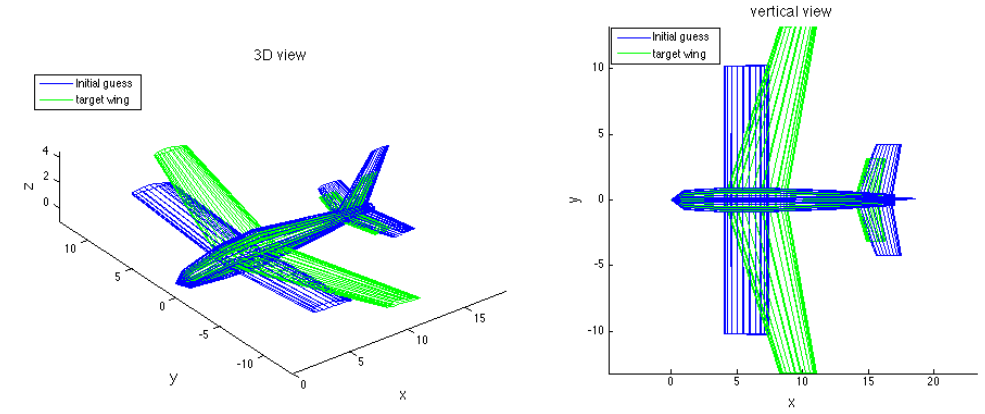
(c) Number of points on horizontal tail



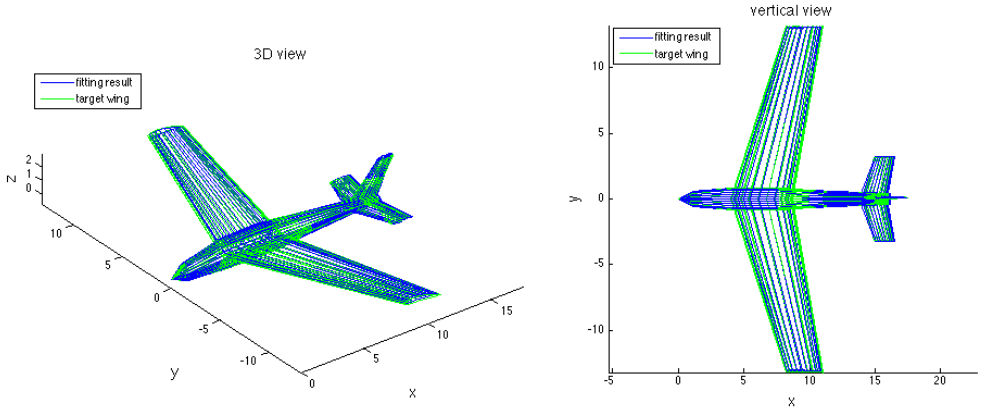
(d) Number of points on vertical tail

Figure 4.38: Number of points associated with each component for 3D glider

The final result adds the classification step to the glider, and the results are shown in Figure 4.39. The RMS of distance from points in cloud to the surface of the glider is shown in Figure 4.40a, where the system converges after 6 cycles. The normalized design parameters for the glider are shown in Figure 4.40b. As can be seen, the generated normalized parameters in the unclassified problem is not as good as in the pre-classified problem. This is because 1% of the points are misclassified. However, the whole technique applied on the cloud of unclassified points still yields an acceptable result.

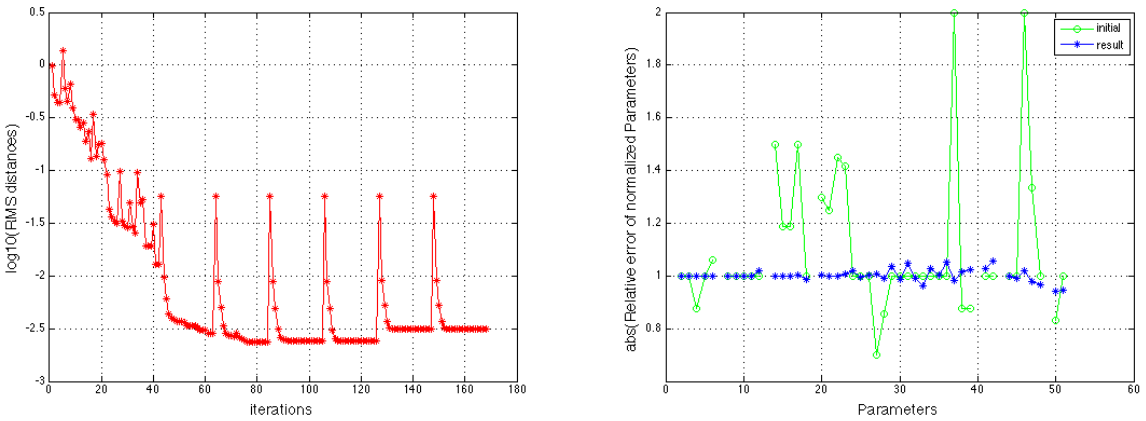


(a) Initial configuration



(b) Final configuration

Figure 4.39: Final fitting result for 3D glider



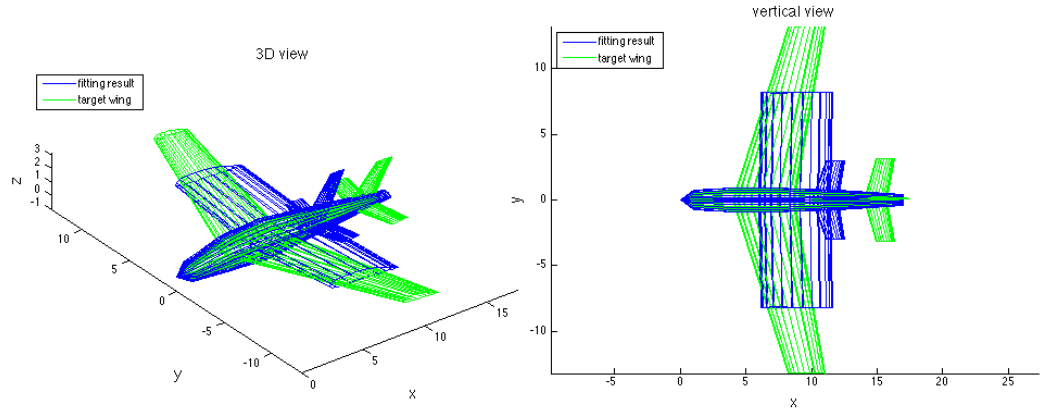
(a) RMS distances

(b) Normalized parameters

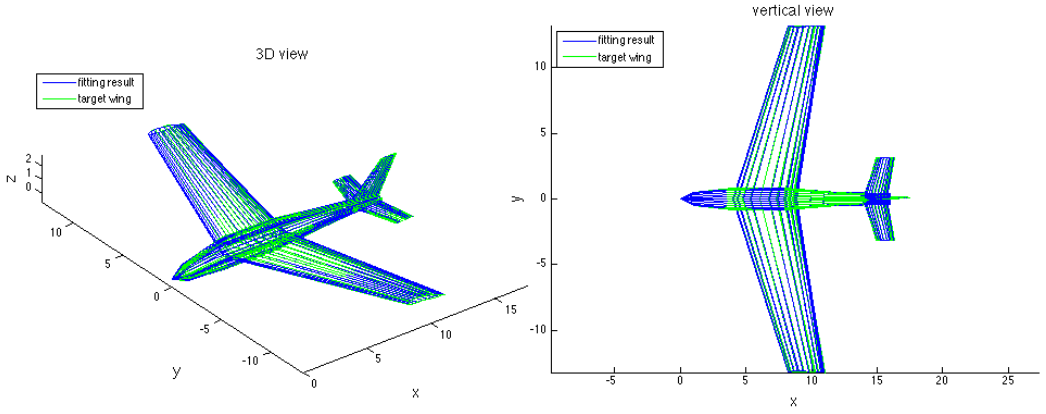
Figure 4.40: RMS distances and normalized parameters for 3D glider

4.5.3 Improvement of ILM Method

For focusing on the effect of ILM algorithm during glider fitting problem, the points in cloud are pre-classified (each point associated to the correct component). The initial guess and fitting result of the test problem are shown in Figure 4.41. The green lines are the target for fitting and the blue lines are the initial guess and fitting result.



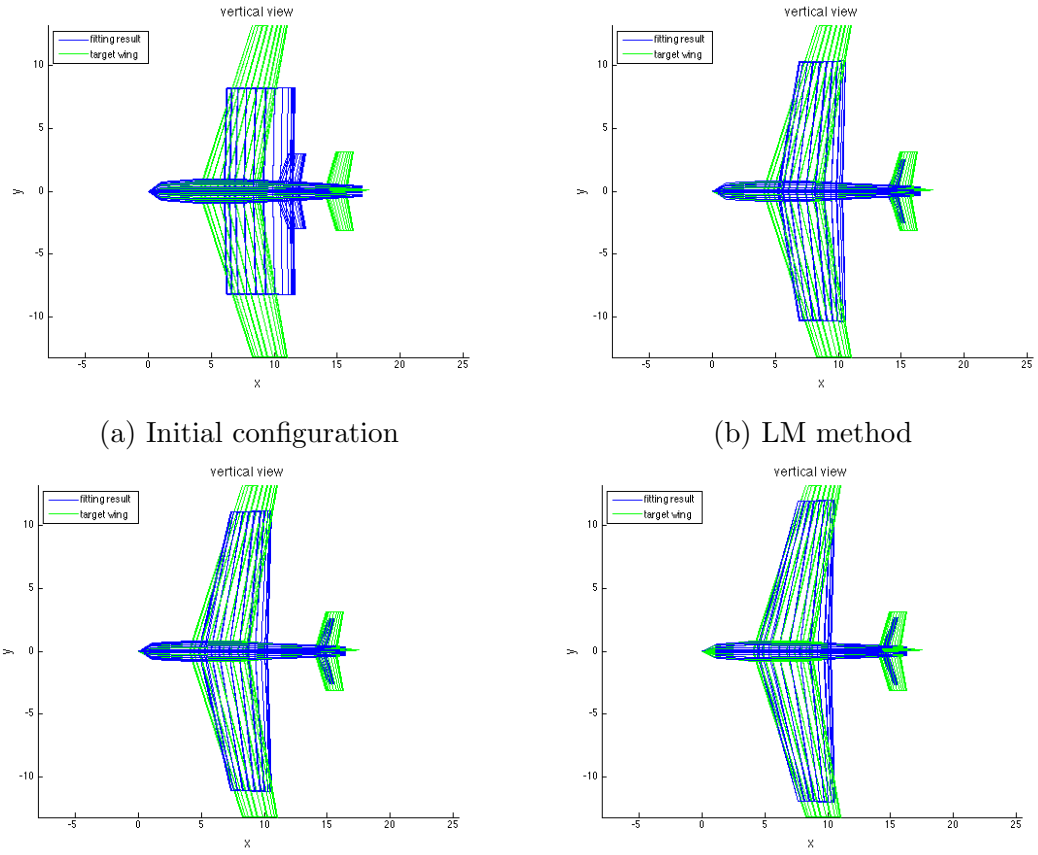
(a) Initial configuration



(b) Final fit configuration

Figure 4.41: Initial and final results for 3D glider (pre-classified)

The fitting results after 10 iterations based on different updating rules are shown in Figure 4.42. The green lines are the target for fitting, and the blue lines are the result of the optimization process. As can be seen, using the simulated annealing idea for updating the iteration result can increase the convergence speed. After 10 iterations, the shape of the parametric model in Figure 4.42d (using ILM method) is closer to the fitting target comparing with the shape in Figure 4.42b or in Figure 4.42c (which corresponds to a scheme in which trial solutions are accepted based upon a random number, as is done in simulated annealing alone).



(a) Initial configuration

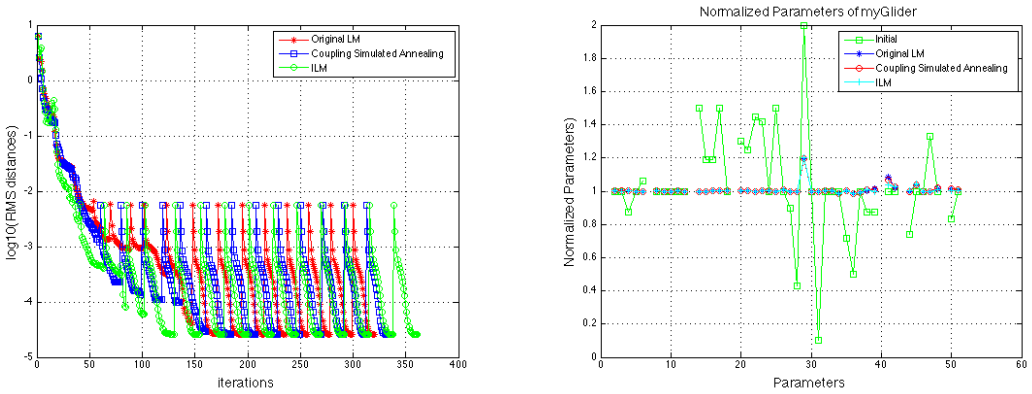
(b) LM method

(c) LM coupling SA method

(d) ILM method

Figure 4.42: Comparison of different variants of LM method after 10 iterations

Figure 4.43a shows the RMS distances between points in the cloud and generated glider. After relaxing the condition of updating the iterations' result, the RMS distance reduced more quickly. It can get the correct geometry model after 5 cycles, as opposed to 8 cycles during using the original LM method. Also, the normalized parameters are all nearly one, which are shown in Figure 4.43b.



(a) RMS distances (b) Normalized parameters

Figure 4.43: RMS distances and normalized parameters for 3D glider

All cases above are calculated with the derivatives by hand, and coded in MATLAB. In order to apply the algorithm to more general geometry configuration generation, one needs a platform that can provide a geometry model easily and calculate the analytical derivatives automatically. Therefore, the Engineering Sketch Pad (ESP) is used as the library for applying the new algorithm. Additionally, all the algorithm will be written in C code when it is integrated into ESP platform. And the running time is 100 times faster than in MATLAB.

Chapter 5

Analysis of Accuracy, Robustness and Efficiency

In this chapter, the algorithm will be used for generating more types of parametric geometry models based on the cloud of points. For applying the algorithm to more general geometry configurations generation, Engineering Sketch Pad (ESP) is used as the platform for the algorithm. ESP is a feature-based solid-modeling system that is web-enabled and can be used with most modern web browsers. In many ways it mirrors the functionality of modern parametric commercial CAD systems. [124]. It can provide a general way for generating the solid geometry model. And it is also be able to compute the sensitivity of the objective function with respect to the driving parameters in a robust and efficient manner.[125].

5.1 ESP Introduction

Within the multi-disciplinary analysis and optimization community, there is a strong need for browser-based tools that provide users with the ability to visualize and interact with complex three-dimensional configurations. This need is particularly acute when the designs involve shape- and/or feature-based optimizations. Described herein is a family of open-sources software products that provide such a capability. At the top is a browser-based system, called the Engineering Sketch Pad (ESP), which provides the user with the ability to interact with a configuration by building and/or modifying the design parameters and feature tree that define the configuration. ESP is built both upon the WebViewer (which is a WebGL-based visualizer for three-dimensional configurations and data) and upon OpenCSM (which is a constructive solid modeler; it in turn is built upon the EGADS and OpenCASCADE systems)[124]. The OpenCSM is the main library that will be used for integrating the fitting algorithm. This will be introduced in the following sections.

5.1.1 Geometry Model Generation (CSM file)

Open Source Constructive Solid Modeler (OpenCSM) is same as the constructive solid modeling (CSG) in modern CAD systems, such a CatiaV5, SolidWorks, and Pro/ENGINEER. In these approaches, a model consists of two types of items:

1. A build recipe (sometimes called a feature tree) that describes the types of and order of operations that one must perform.

2. A set of design parameters that influence the exact shape of objects created (and sometimes which part of the feature tree should be executed).

Comparing with the traditional CAD software, OpenCSM:

1. provides the easily coding format `.csm` file for describing the solid geometry models.
2. primitives can be pre-defined or defined by the user-self.
3. computes the derivatives/sensitivities analytically.
4. can be modified based on user's problem (open source)

The `.csm` file and the sensitivities calculation are the two key features that will be used in the fitting algorithm. The `.csm` file is discussed in this section. The sensitivities calculation method will be introduced in the next section.

`.csm` file will be used as the input for the algorithm of generating parametric models. This file can provide parametric geometry model (build recipe) and the initial guess of design parameters. For each iteration of the optimization process, the new design parameters are obtained and updated into the `.csm` file for rebuilding the new solid geometry model.

5.1.2 Analytical Sensitivity Generation

OpenCSM provides a pair of complementary techniques for computing configuration sensitivities directly on parametric, CAD-based geometries. This technique computes the configuration sensitivity analytically by differentiating the geometry-generating process. [125]

This method needs to have access to the processes used to generate the various operations; this includes the generation of the primitives as well as construction operations such as filleting. Analytical derivatives can also be calculated if the construction process for the primitives can be reverse engineered, as was done here in OpenCSM.

When computing analytic sensitivities on Faces, the first step is to determine the primitive that originally created that Face. Once the correspondence between the root (final) and primitive (original) Faces is known, the point at which the sensitivity is desired, say $\vec{x}_{root}$ is transformed into $\vec{x}_{prim}$ by walking up the feature tree from the root to the element that created the Face, and applying the inverse of the transformations that were traversed.

5.1.3 Integrate the New Algorithm

The algorithm of generating parametric models is coded in C for working on the ESP platform. The program is named `matchCSM.c`. Because the program is built upon the OpenCSM and EGADS, several APIs [126] need to be used during the programming process. The APIs used in the `matchCSM` are listed below:

- `ocsmLoad` – read a `.csm` file and create a model (feature tree branches)
- `ocsmBuild` – execute the feature tree and create a group of Bodies
- `ocsmGetXYZ` – get the coordinates of the points on geometry model
- `ocsmGetValu` – get the definition and value of a design parameter
- `ocsmSetValu` – set the definition and value of a design parameter
- `ocsmGetVel` – get the sensitivity of a design parameter
- `EG_evaluate` – get the sensitivity of the parametric coordinates
- `EG_attributeRet` – get an attribute associated with a particular branch

5.2 Generalization and Accuracy

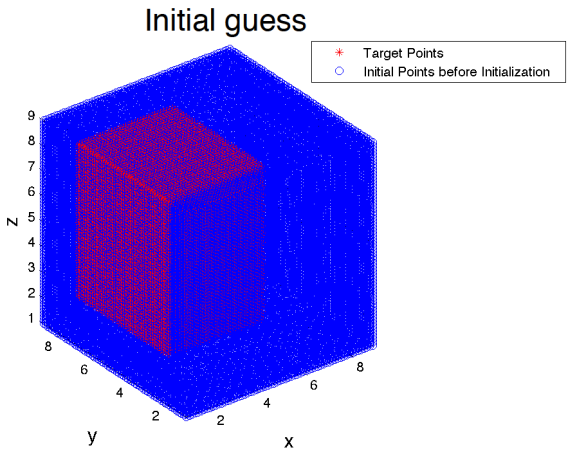
In this section, the `matchCSM` is used for fitting 6 different geometry models for testing the generalization of the program. The key standard for evaluating the accuracy is the result of normalized design parameters. If they are equal to 1 or almost to 1, that means the accurate results can be obtained by the program.

The Figure 5.1 - 5.6 are the fitting cases regarding box, rotated box, fuselage, wing, glider and plane (with engines). The red points are the cloud of points (targets). The blue points in Figure (a)s are the configurations generated by the initial guess design parameters. The blue points in Figure (b)s are the configurations generated by the fitting results. Figure (c)s show the RMS of the distances between the points to the configurations during the optimization process. Figure (d)s are the normalized results of design parameters for each geometry model. As shown in these figures, for the single-component configuration, the normalized design parameters are equal to 1 at the end of the iterations. On the other hand, for the multiple-component geometries, results of the glider fitting and plane fitting are acceptable (normalized design parameters are almost = 1). The original design parameters and fitting results are shown in below:

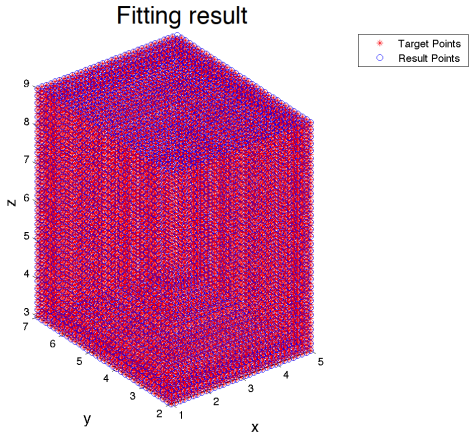
5.2.1 Box Testing Case

Table 5.1: Design Parameters of the Box (in ESP)

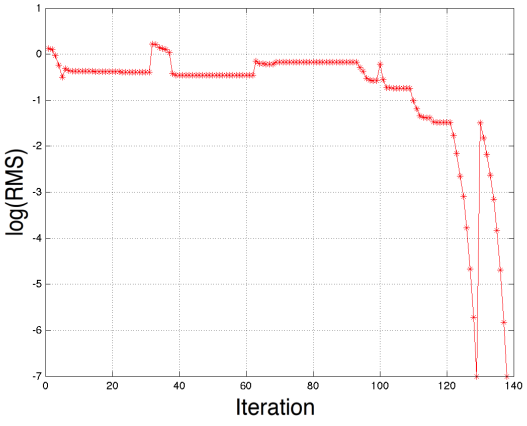
X Location	Y Location	Z Location	X Length	Y Length	Z Length
1	2	3	4	5	6



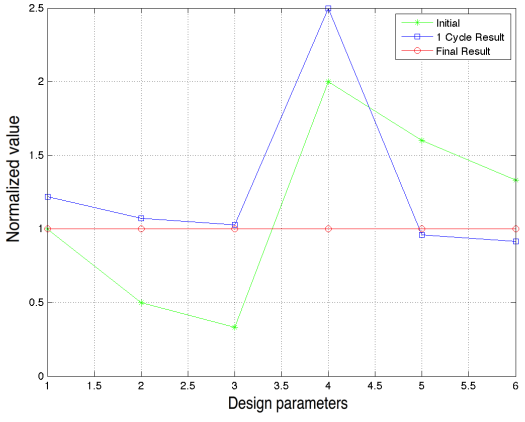
(a) Initial guess and target



(b) Fitting result



(c) RMS of distance



(d) Normalized design parameters

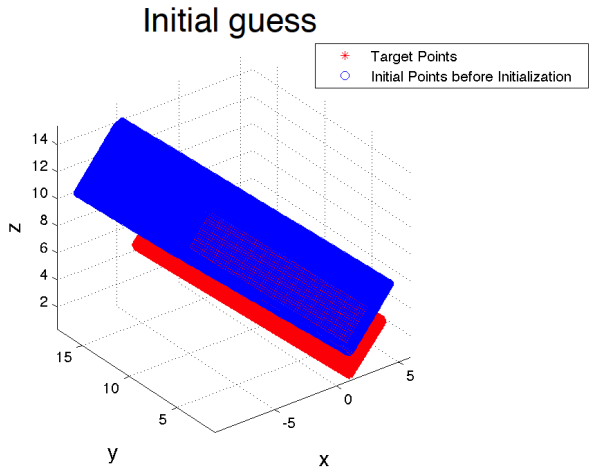
Figure 5.1: Box fitting results

As shown in the above, the correct design parameters can be obtained after 130 iterations for box fitting problem.

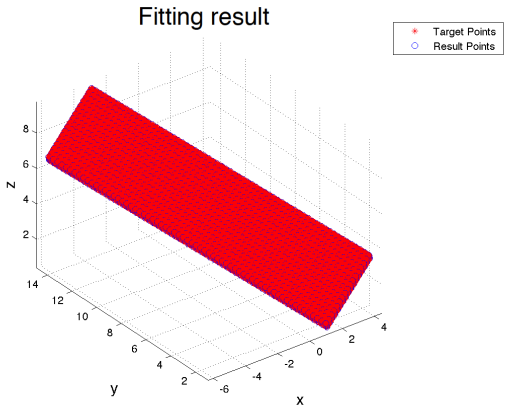
5.2.2 Rotated Box Testing Case

Table 5.2: Design Parameters of the Rotated Box (in ESP)

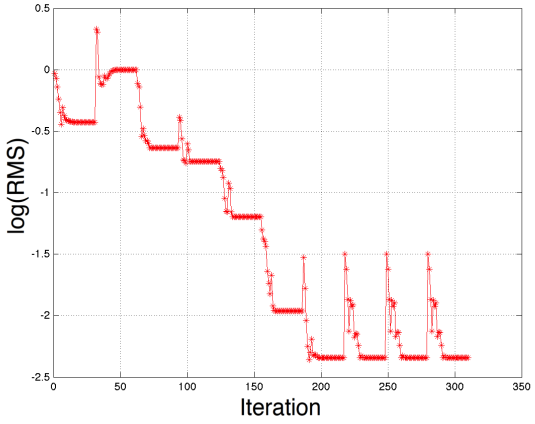
X Location	Y Location	Z Location	X Length	Y Length	Z Length	X Angle	Y Angle	Z Angle
1	1	1	1	16	4	30	40	50



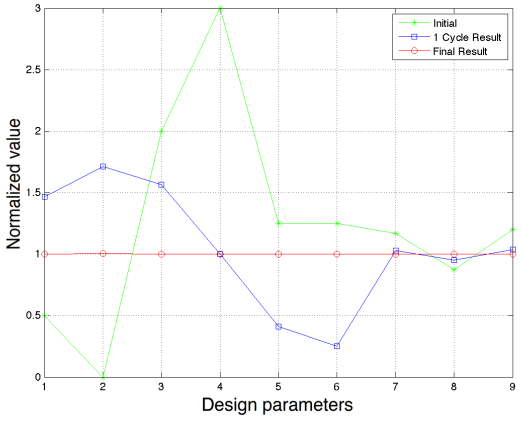
(a) Initial guess and target



(b) Fitting result



(c) RMS of distance



(d) Normalized design parameters

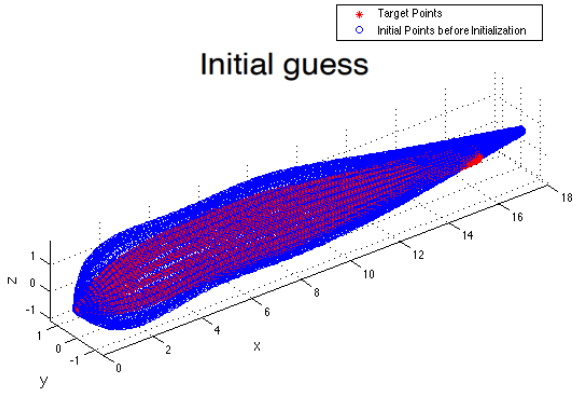
Figure 5.2: Rotated box fitting results

Due to the rotation angles are added into design parameters, the correct correct values can be obtained after 310 iterations for rotated box fitting problem.

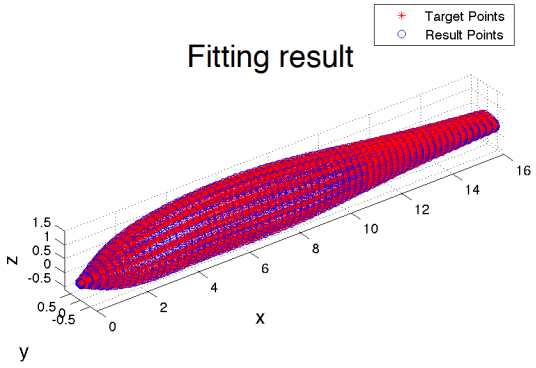
5.2.3 Fuselage Testing Case

Table 5.3: Design Parameters of the Fuselage (in ESP)

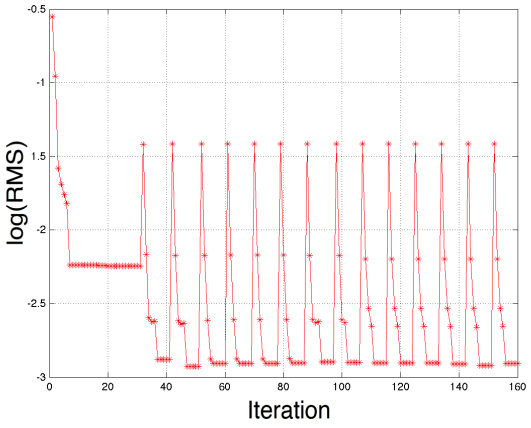
	Section1	Section2	Section3	Section4	Section5	Section6
X Location	0	1	4	8	12	16
Z Location	0	0.1	0.4	0.4	0.3	0.2
Width	0	1	1.6	1.6	1	0.8
Height	0	1	2	2	1.2	0.4



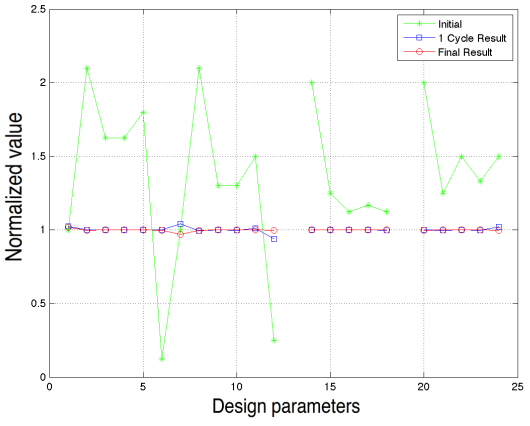
(a) Initial guess and target



(b) Fitting result



(c) RMS of distance



(d) Normalized design parameters

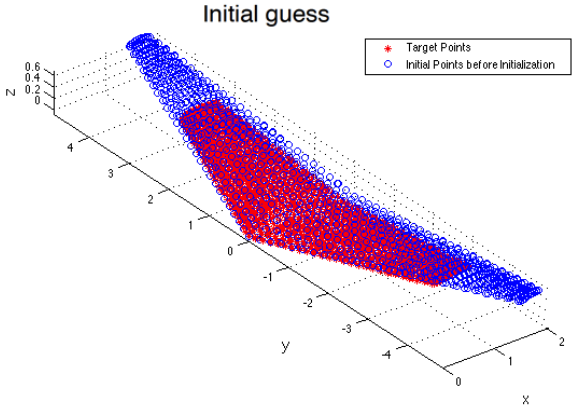
Figure 5.3: Fuselage fitting results

The fuselage model is a blend of 6 super-ellipses sections, the correct design parameters can be obtained after 15 cycles and total 160 iterations. In the figure of normalized design parameters, the gap between two points is due to the original design parameters at that point is 0.

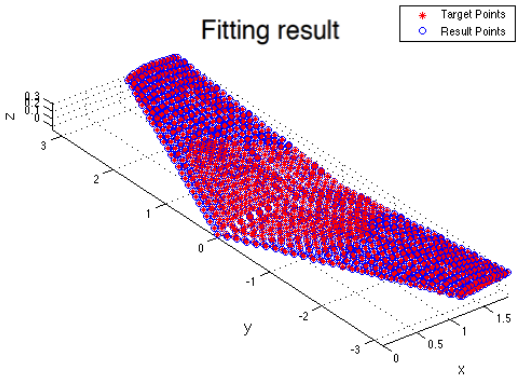
5.2.4 Wing Testing Case

Table 5.4: Design Parameters of the Wing (in ESP)

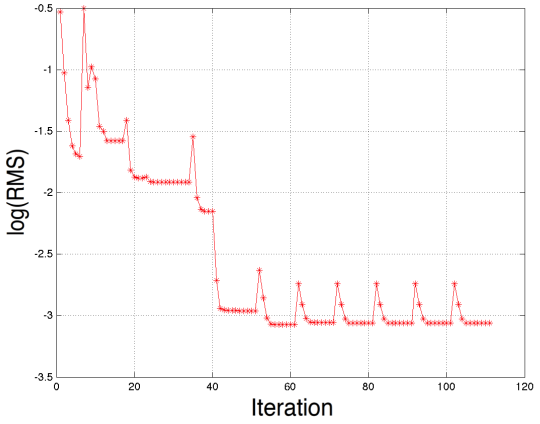
Thick	Camber	Area	Aspect	Taper	Sweep	Twist	Dihedral
0.12	0.04	100	7	0.6	10	-5	5



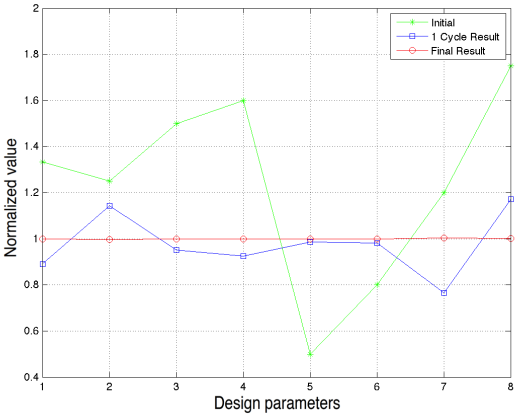
(a) Initial guess and target



(b) Fitting result



(c) RMS of distance



(d) Normalized design parameters

Figure 5.4: Wing fitting results

The wing model is ruled by the root and tip, the correct design parameters can be obtained after 11 cycles and total 110 iterations.

5.2.5 Glider Testing Case

Table 5.5: Design Parameters of the Fuselage in Glider (in ESP)

	Section1	Section2	Section3	Section4	Section5	Section6
X Location	0	1	4	8	12	16
Z Location	0	0.1	0.4	0.4	0.3	0.2
Width	0	1	1.6	1.6	1	0.8
Height	0	1	2	2	1.2	0.4

Table 5.6: Design Parameters of the Wing in Glider (in ESP)

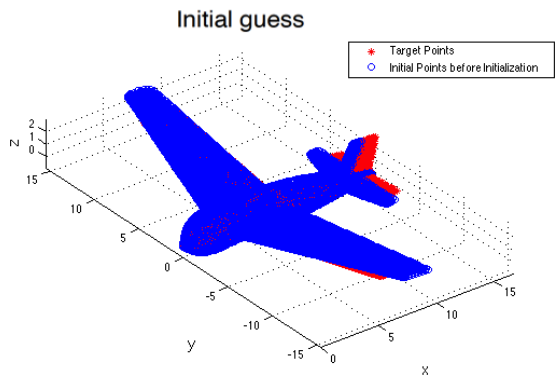
X Location	Z Location	Thick	Camber	Area	Aspect	Taper	Sweep	Twist	Dihedral
4	0.2	0.12	0.04	100	7	0.6	10	-5	5

Table 5.7: Design Parameters of the Horizontal Tail in Glider (in ESP)

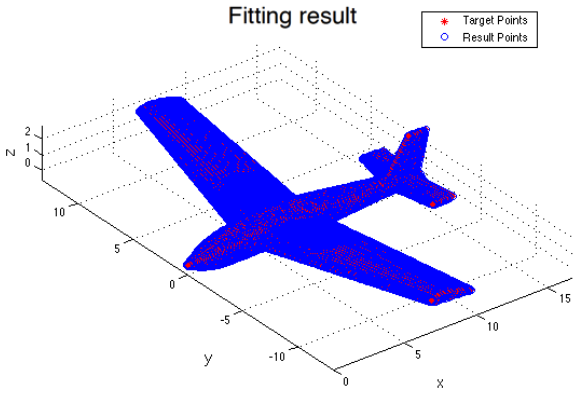
X Location	Z Location	Thick	Camber	Area	Aspect	Taper	Sweep	Twist	Dihedral
14	0.2	0.1	0	10	4	0.8	10	0	0

Table 5.8: Design Parameters of the Vertical Tail in Glider (in ESP)

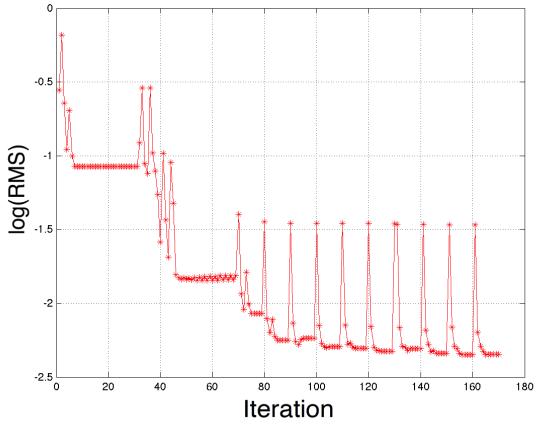
X Location	Z Location	Thick	Camber	Area	Aspect	Taper	Sweep
13.5	0.1	0.1	0	10	3	0.5	30



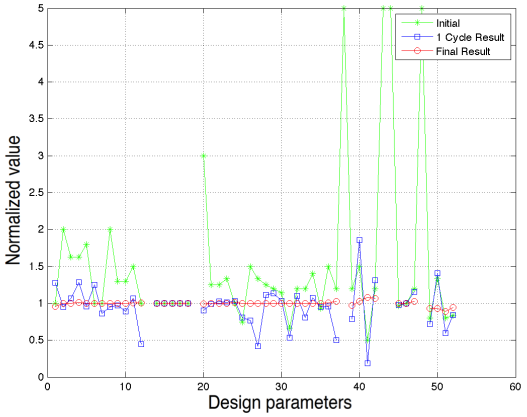
(a) Initial guess and target



(b) Fitting result



(c) RMS of distance



(d) Normalized design parameters

Figure 5.5: Glider fitting results

The glider model is composed by the fuselage, wing, vertical tail and horizontal tail. Because the glider model is multiple components and more complicate structure, the approximate correct design parameters can be obtained after 15 cycles and total 170 iterations.

5.2.6 Plane Testing Case

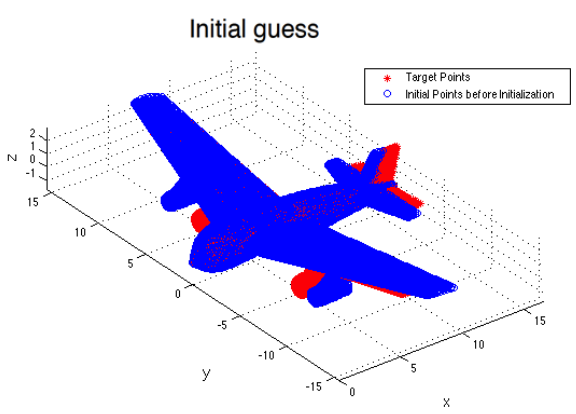
The design parameters of the plane are same as the glider's except added two more components on the glider. They are:

Table 5.9: Design Parameters of the Engine in Plane (in ESP)

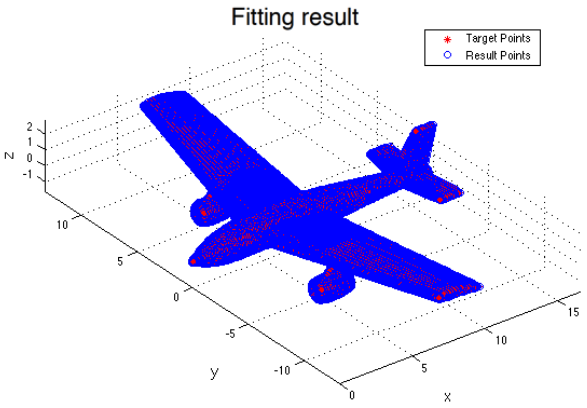
X Location	Diameter	Length	Thickness	Camber	Percent of Span
0.5	1	4	0.05	0.04	0.4

Table 5.10: Design Parameters of the Strut (connection between engine and wing) in Plane (in ESP)

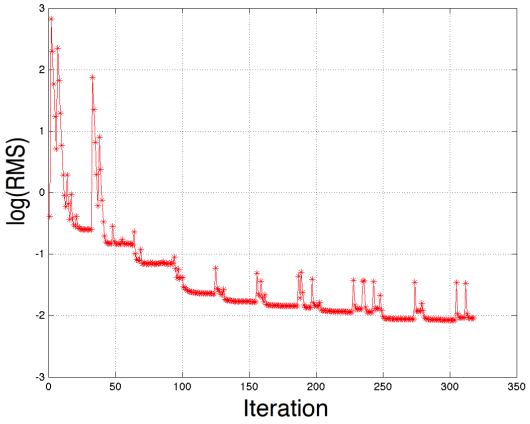
X Location	Length	Thickness	Sweep
0.4	1	0.25	45



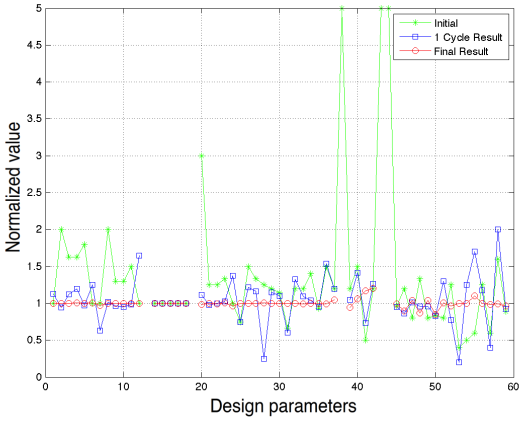
(a) Initial guess and target



(b) Fitting result



(c) RMS of distance



(d) Normalized design parameters

Figure 5.6: Plane fitting results

The plane model is composed by the glider, engines and struts (connections between wing and engines) . Because the plane model contains more small parts, the approximate correct design parameters can be obtained after 15 cycles and total 320 iterations.

5.3 Robustness

In this section, the robustness for the different initial guess is tested when using the `matchCSM` for the fuselage fitting problem. 6 sets of different design parameters are assigned to the parametric fuselage model, as in Table 5.12 - 5.17. There are diverse initial guesses in their 6 sets. They included: initial guess bigger than the target (Figure5.8) , initial guess smaller then the target (Figure5.9), the initial guess cross the target (Figure5.10), the initial guess fluctuates around the target (Figure5.11), initial guess rotated in one direction (Figure5.12)and initial guess rotated in 3 directions (Figure5.13). As shown in the Figures, the normalized design parameters are all equal or almost equal to 1. This means the program is robust for different types of initial guesses.

The original fuselage model is shown in Figure5.7, and the design parameters are shown in Table5.11.

Table 5.11: Design Parameters of Rotated Fuselage

	Section1	Section2	Section3	Section4	Section5	Section6
X Location	0	1	4	8	12	16
Z Location	0	0.1	0.4	0.4	0.3	0.2
Width	0	1	1.6	1.6	1	0.8
Height	0	1	2	2	1.2	0.4
X Rotation			0			
Y Rotation			0			
Z Rotation			0			

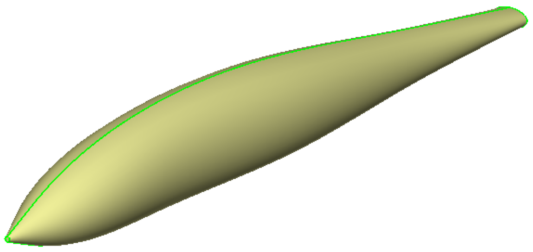
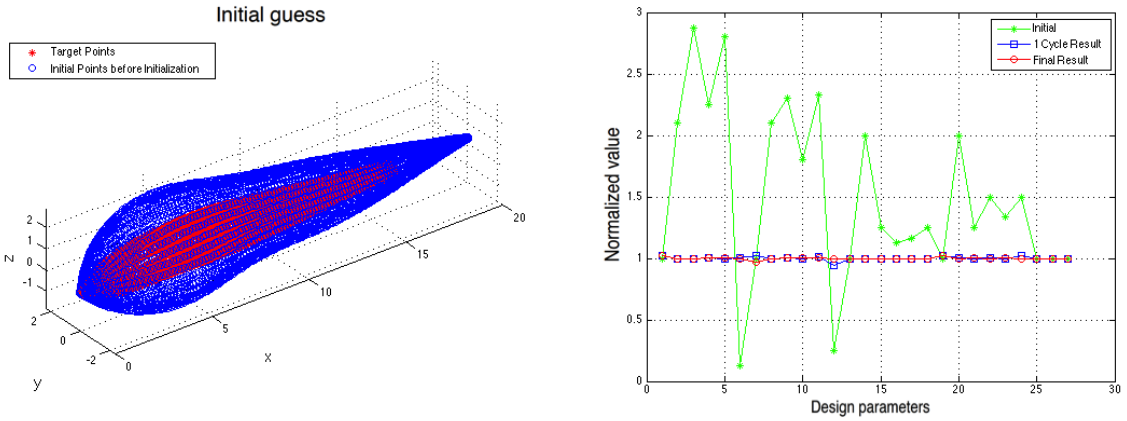


Figure 5.7: Original fuselage model

5.3.1 Initial Guess Larger than the Target Configuration

Table 5.12: 1st Set of Initial Design Parameters for Rotated Fuselage

	Section1	Section1	Section3	Section4	Section5	Section6
X Location	0	2	5	9	14	20
Z Location	0	0.2	0.5	0.6	0.4	0.3
Width	0.1	2.1	4.6	3.6	2.8	0.1
Height	0.1	2.1	4.6	3.6	2.8	0.1
X Rotation			0			
Y Rotation			0			
Z Rotation			0			



(a) Initial guess and target (b) Normalized design parameters

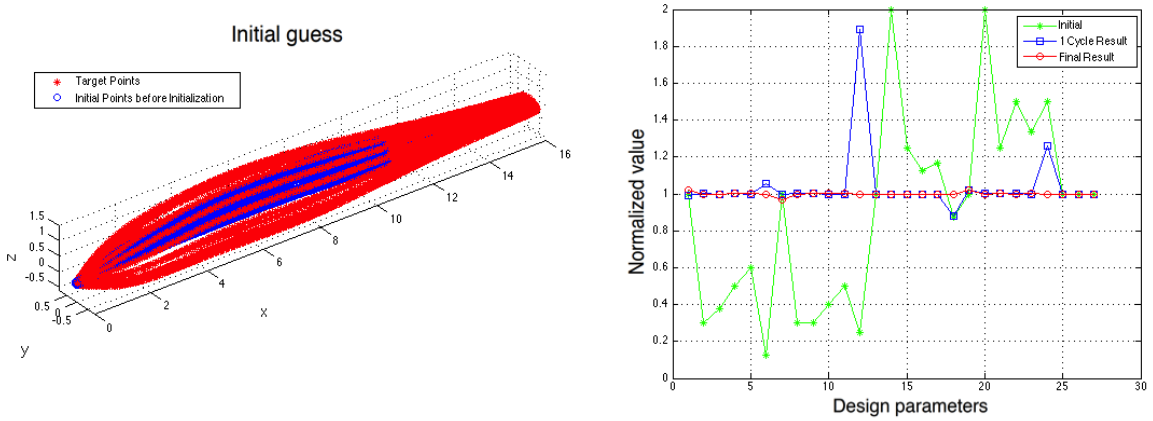
Figure 5.8: Fuselage fitting results based on the 1st initial guess

As shown in the above, starting from the large initial guess, the original design parameters can be obtained after using `matchCSM` as fitting technique.

5.3.2 Initial Guess Smaller than the Target Configuration

Table 5.13: 2nd Initial Design Parameters of Rotated Fuselage

	Section1	Section1	Section3	Section4	Section5	Section6
X Location	0	2	5	9	14	14
Z Location	0	0.2	0.5	0.6	0.4	0.3
Width	0.1	0.3	0.6	0.8	0.6	0.1
Height	0.1	0.3	0.6	0.8	0.6	0.1
X Rotation			0			
Y Rotation			0			
Z Rotation			0			



(a) Initial guess and target (b) Normalized design parameters

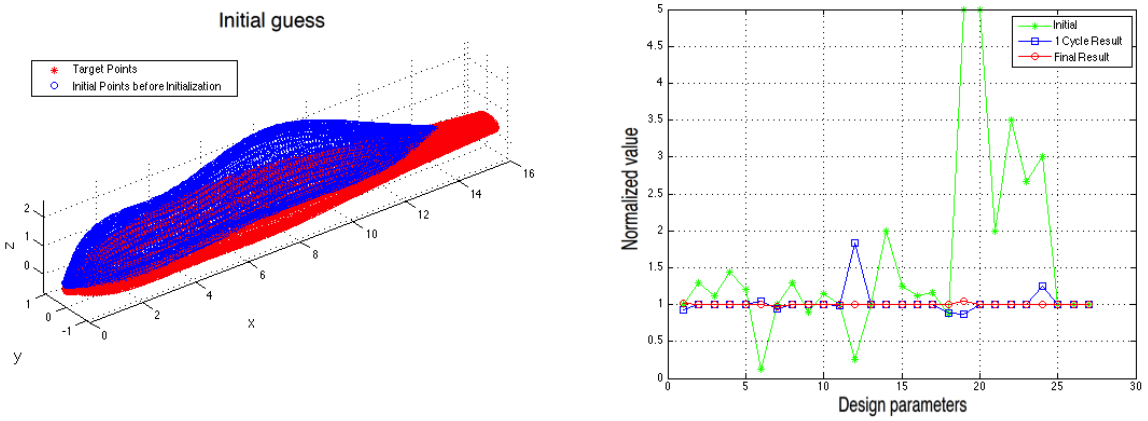
Figure 5.9: Fuselage fitting results based on the 2nd initial guess

As shown in the above, starting from the small initial guess, the original design parameters can be obtained after using `matchCSM` as fitting technique.

5.3.3 Initial Guess Cross the Target Configuration

Table 5.14: 3rd Initial Design Parameters of Rotated Fuselage

	Section1	Section1	Section3	Section4	Section5	Section6
X Location	0	2	5	9	14	20
Z Location	0.2	0.6	0.8	1.4	0.8	0.6
Width	0.1	1.3	1.8	2.3	1.2	0.1
Height	0.1	1.3	1.8	2.3	1.2	0.1
X Rotation			0			
Y Rotation			0			
Z Rotation			0			



(a) Initial guess and target (b) Normalized design parameters

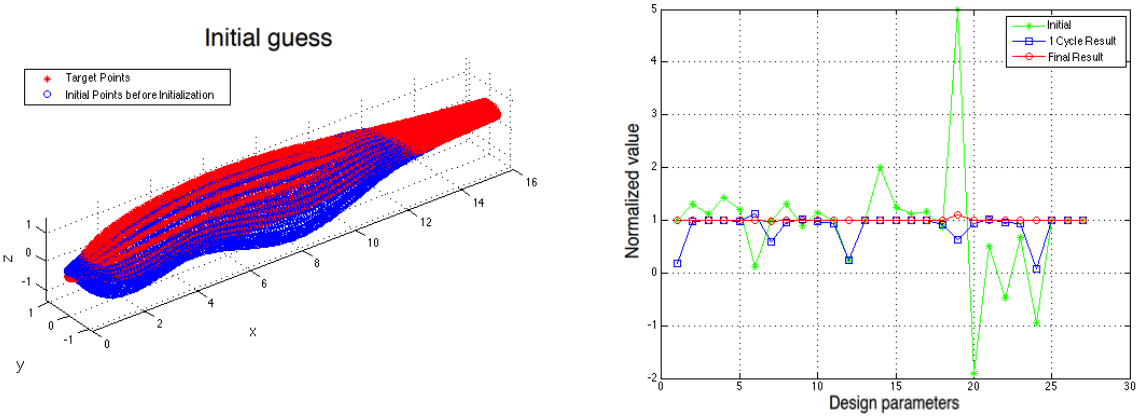
Figure 5.10: Fuselage fitting results based on the 3rd initial guess

As shown in the above, starting from the initial guess configuration cross the cloud of points, the original design parameters can be obtained after using `matchCSM` as fitting technique.

5.3.4 Initial Guess Fluctuates around the Target Configuration

Table 5.15: 4th Initial Design Parameters of Rotated Fuselage

	Section1	Section1	Section3	Section4	Section5	Section6
X Location	0	2	5	9	14	20
Z Location	0.2	-0.2	0.2	-0.2	0.2	-0.2
Width	0.1	1.3	1.8	2.3	1.2	0.1
Height	0.1	1.3	1.8	2.3	1.2	0.1
X Rotation			0			
Y Rotation			0			
Z Rotation			0			



(a) Initial guess and target (b) Normalized design parameters

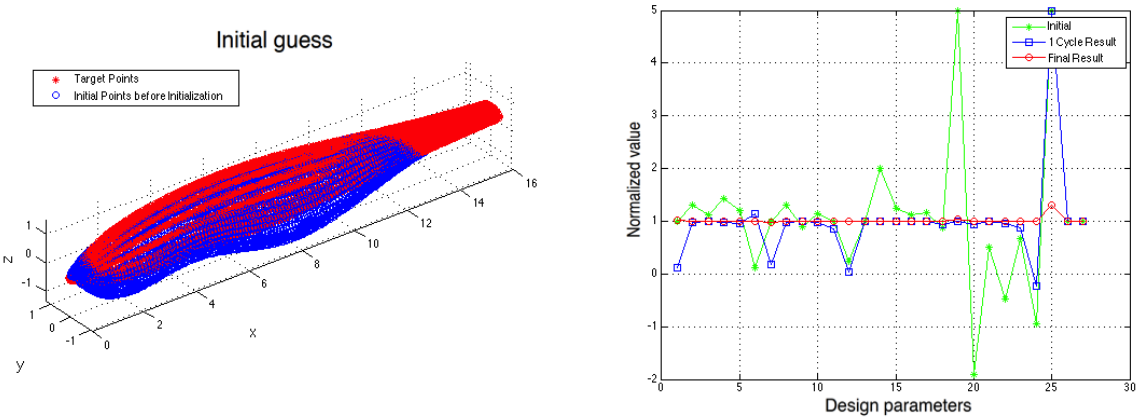
Figure 5.11: Fuselage fitting results based on the 4th initial guess

As shown in the above, starting from the initial guess configuration fluctuates the cloud of points, the original design parameters can be obtained after using `matchCSM` as fitting technique.

5.3.5 Initial Guess Rotated in 1 Direction

Table 5.16: 5th Initial Design Parameters of Rotated Fuselage

	Section1	Section1	Section3	Section4	Section5	Section6
X Location	0	2	5	9	14	20
Z Location	0.2	-0.2	0.2	-0.2	0.2	-0.2
Width	0.1	1.3	1.8	2.3	1.2	0.1
Height	0.1	1.3	1.8	2.3	1.2	0.1
X Rotation	20					
Y Rotation	0					
Z Rotation	0					



(a) Initial guess and target (b) Normalized design parameters

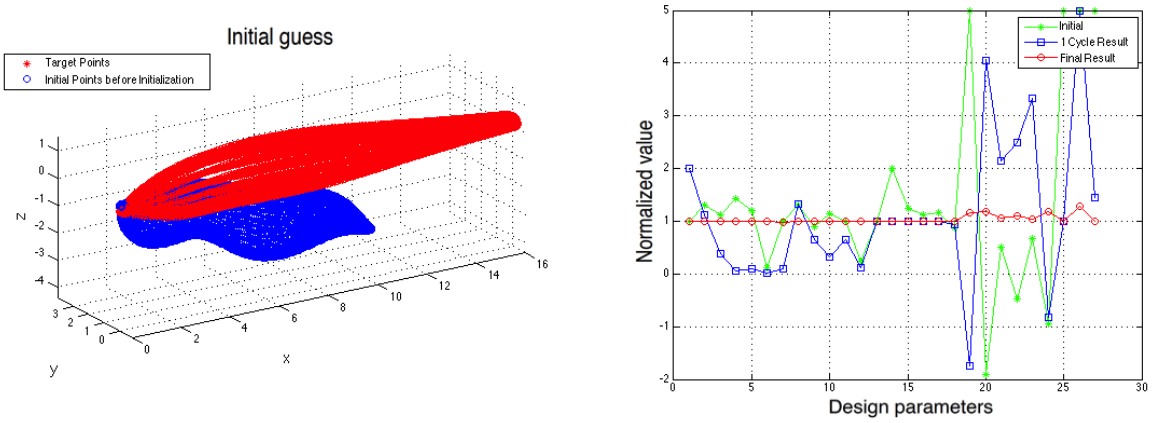
Figure 5.12: Fuselage fitting results based on the 5th initial guess

As shown in the above, starting from the initial guess configuration rotated in 1 direction, the original design parameters can be obtained after using `matchCSM` as fitting technique.

5.3.6 Initial Guess Rotated in 6 Direction

Table 5.17: 6th Initial Design Parameters of Rotated Fuselage

	Section1	Section1	Section3	Section4	Section5	Section6
X Location	0	2	5	9	14	20
Z Location	0.2	-0.2	0.2	-0.2	0.2	-0.2
Width	0.1	1.3	1.8	2.3	1.2	0.1
Height	0.1	1.3	1.8	2.3	1.2	0.1
X Rotation			20			
Y Rotation			20			
Z Rotation			20			



(a) Initial guess and target (b) Normalized design parameters

Figure 5.13: Fuselage fitting results based on the 6th initial guess

As shown in the above, starting from the initial guess configuration rotated in 3 directions, the original design parameters can be obtained after using `matchCSM` as fitting technique.

5.4 Efficiency

In this section, the efficiency of `matchCSM` is tested in 3 ways. First, one tests the general running time for different types of geometry models. Second, using the wing as the geometry model, one tests the running time based on different number of faces. Third, also using the wing as the geometry model, one tests the running time based on different number of points.

5.4.1 Complexity of the Algorithm

For analyzing the complexity of the program, we recall the sparse technique in Chapter 3, section 3.3.2. In this Chapter, the Hession matrix for 3D problem is shown in Figure 5.14. After using sparse technique, the space of storing the Jacobian matrix $O(m)$, where m is the number of points. For the arithmetic of sparse Jacobian matrix, there are $2 \times n \times n \times m$ plus and multiply operations in block A. There are $2 \times n \times m$ multiply operations in block B_1, B_2, C_1 and C_2. There are $2 \times m$ multiply operations in block $D_1 - D_4$. Therefore, the complexity of arithmetic is $O(m)$ due to $n << m$.

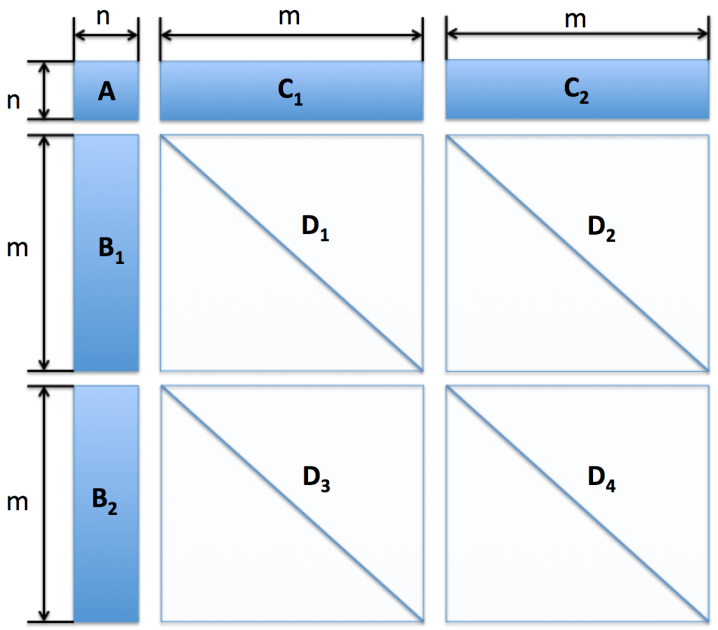


Figure 5.14: Hession matrix structure for 3D problem

5.4.2 Time Analysis for Different Geometry Configurations

In this section, the running times of different types geometry models generation are recorded. For all the tested cases, the maximum number of iterations `IterMax` is set as 30. The maximum number of `cycle` is 15. If the final RMS is not reduced in 2 cycles, the process will be stopped. The coefficient of classification `cof` is set as 1.5. From Table 5.18, the running time for each case is listed. The `Time(optimizer)` means the time ignored the geometry building time. `Per ILM` means the time for calculating each iteration function. `Per Initialization` means the time for each initialization/classification process. `Per Derivative` means the time for generating the derivative of each design parameters.

The total running time of the multiple-component configurations is much more than the time of single-component configurations. The reason is that the generation of multiple-component configurations involves the `UNION` Boolean operation in the `ocsmBuild` function. The `UNION` operation is provided by the OpenCASCADE library, and is very time

consuming. Because the geometry building process is not the key point in the geometry fitting research, the time ignored in the `ocsmBuild` is also recorded for analyzing the efficiency of the fitting algorithm. If one focus on the optimization algorithm (ignoring the `ocsmBuild` time), the running time of generating multiple-component configurations is reduced to less than 10% of the original running time.

Table 5.18: Running Time of Generating Different Parametric Models

Geometry model	Box	Rotated Box	Fuselage	Wing	Glider	Plane
Number of points	10758	44358	6114	1731	22930	30516
Cycle	7	10	15	11	15	15
Iterations	138	310	146	103	207	313
Time/iter	0.05	0.23	0.86	0.67	45.93	238.34
Time(optimizer)/iter	0.03	0.19	0.55	0.04	4.75	7.81
Total time	22.69	97.64	141.69	75.92	10148.37	77034.54
Total time (optimizer)	19.80	85.13	94.75	8.83	1434.91	4334.24
Per ILM	0.02	0.12	0.09	0.01	2.49	3.55
Per Initialization	0.11	0.08	0.02	0.02	0.35	0.31
Per Derivative	0.01	0.06	0.45	0.04	2.23	4.22
Per Build	0.02	0.04	0.32	0.65	42.09	232.27

The plane model is the most complicate geometry model being fitted in this research. To sum up, the plane fitting process can be finished in 21 hours as total time for 3×10^5 points in the cloud (by `matchCSM` of a MacBook Pro). The core optimization algorithm (not count geometry building time) only takes 70 minutes in it.

From Figure 5.15, there is the running time regarding fitting different type configurations. As shown in the figures, the more complex the geometry model, the more time needs to

be spent on the ILM function and generating the Jacobian matrix. In the first 2 cases (box fitting), the time of geometry building takes 10% of the total running time. The time ignoring the "build time" almost can be seen as the running time of the optimization algorithm. For the fuselage case, the optimizer time takes 70% of the total running time. For the wing case, the optimizer time takes 10% of the total running time, because the wing model (`.csm` file) involves the internal calculations for generating the geometry model (like calculating span based on the area and aspect ratio). Therefore, the more complex the geometry model, the more time needs to be spent on the geometry build. In the glider case, the total running time is reduced 85% after ignoring the build time. For the plane case, the time is reduced 93%.

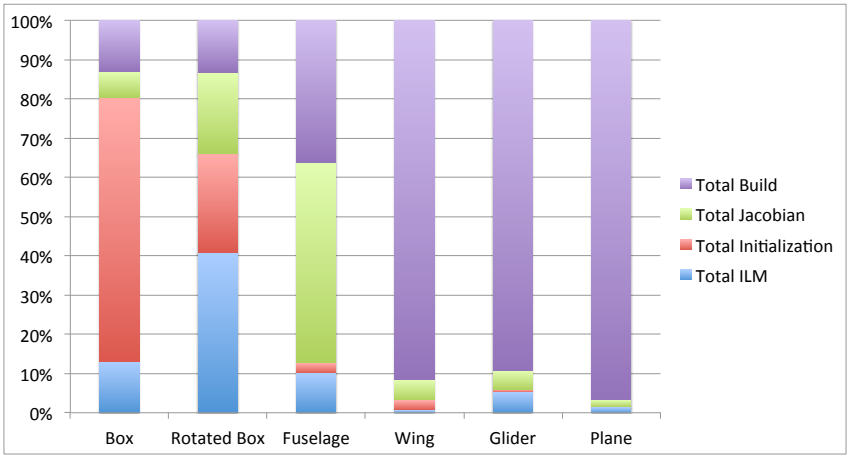
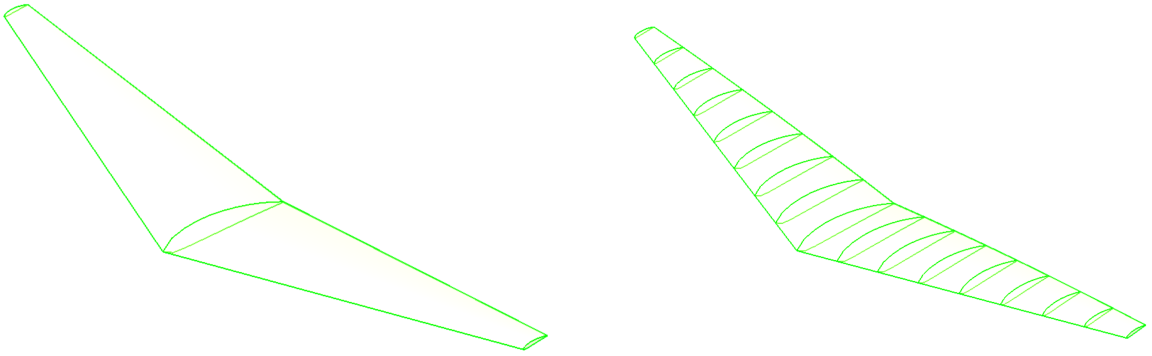


Figure 5.15: Running time distribution for different configurations

5.4.3 Test 8 Different Number of Design Parameters

In this section, the wing is used as the fitting geometry model. We are focusing on the running time changed with the number of design parameters.

In ESP platform, the number of design parameters not only impacts the running time of each iteration function (due to the different different n in Jacobian matrix). As shown in Figure 5.16, there are the 8 design parameters model and 64 design parameters (each section on the wing has a set design parameters) model. During generating the wing model in Figure 5.16b, al l the faces will be blend together in order to making sure the number of faces for this two models are same. The geometry structure (shape) of these cases are the same except the number of design parameters. The results of time is listed in Table 5.19.



(a) 8 design parameters wing (b) 64 design parameters wing

Figure 5.16: Different number of design parameters for the wing generation

Table 5.19: Running Time for Wings in Different Number of Design Parameters (DPs)

Number of points	1731							
Number of faces	8							
Number of DPs	8	16	24	32	40	48	56	64
Cycle	10	10	10	8	9	9	10	10
Iterations	131	127	125	144	177	175	243	266
Time/iter	0.69	1.06	1.42	1.81	2.16	2.49	2.91	3.33
Time(optimizer)/iter	0.04	0.06	0.07	0.07	0.08	0.09	0.11	0.11
Total time	98.79	99.65	103.26	109.75	117.16	124.93	145.36	157.38
Total time (optimizer)	10.79	15.54	17.06	18.82	27.42	30.17	42.19	50.21
Per ILM	0.005	0.009	0.014	0.019	0.024	0.030	0.034	0.041
Per Initialization	0.0128	0.0129	0.0128	0.0127	0.0128	0.0129	0.0128	0.0128
Per Derivative	0.0046	0.0045	0.0045	0.0044	0.0045	0.0046	0.0045	0.0045
Per Build	0.41	0.41	0.41	0.42	0.41	0.42	0.41	0.41

As shown in the Figure 5.17, the vertical axis is the normalized time T_N (normalized time is the ratio of current time value to the first time value, the formula is $T_{Ni} = T_i/T_1$). The time per ILM function increased linearly with the number of design parameters due to the more columns in Jacobian matrix. Here, sparse techniques developed in Chapter 3 reduced the complexity of matrix (and linear equations) calculation into $O(n)$. The time per initialization/classification, the time for generating sensitives for each design parameter and the time for building the geometry model are keep constants.

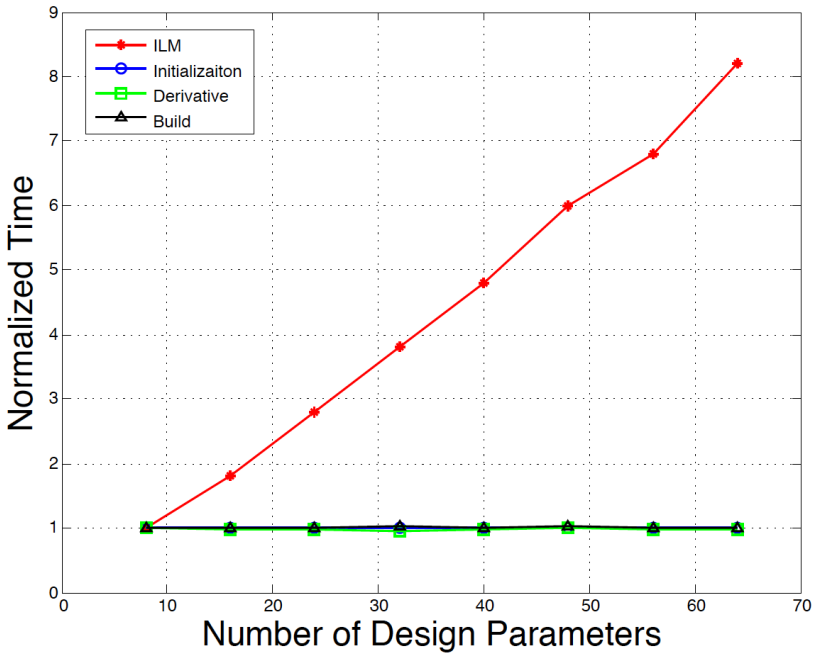


Figure 5.17: Running time analysis based on different number of design parameters

5.4.4 Test 8 Different Number of Faces

Because the OpenCSM constitutes the geometry models based on faces, the points in cloud should be classified to different faces during the classifying process. And the "ocsmGetVel" also generates the sensitivities of design parameters based on different faces. In this section, the wing is used as the fitting geometry model. We are focusing on the running time changed with the number of faces. The design parameters and geometry structure are the same except the number of faces. The results of time is listed in Table 5.20.

Table 5.20: Running Time for Wings in Different Number of Faces

Number of points	1731							
Number of DPs	8							
Number of faces	8	14	20	26	32	38	44	50
Cycle	10	10	10	8	9	9	10	10
Iterations	131	124	125	144	177	175	243	266
Time/iter	0.69	1.05	1.41	1.78	2.14	2.46	2.88	3.28
Time(optimizer)/iter	0.04	0.05	0.05	0.05	0.06	0.06	0.07	0.07
Total time	98.79	155.37	192.26	269.18	394.80	447.56	721.92	897.81
Total time (optimizer)	10.79	14.23	16.06	16.95	24.06	25.80	35.15	40.63
Per ILM	0.005	0.006	0.006	0.005	0.005	0.005	0.005	0.005
Per Initialization	0.0128	0.0131	0.0133	0.0134	0.0143	0.0157	0.0155	0.0163
Per Derivative	0.0045	0.0054	0.0056	0.0057	0.0067	0.0068	0.0072	0.0079
Per Build	0.41	0.67	0.90	1.13	1.36	1.57	1.83	2.09

As shown in the Figure 5.18, the time per ILM function keeps constant and does not change with the number of faces. However, the time per initialization/classification, time for generating sensitives for each design parameter and the time for each build function are increased linearly with the number of faces.

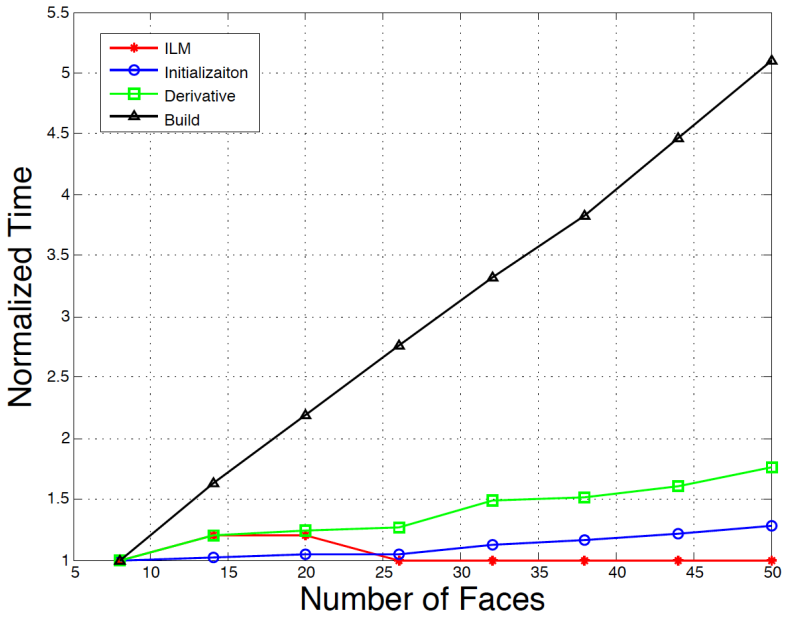


Figure 5.18: Running time analysis based on different number of faces

5.4.5 Test 8 Different Number of Points in Cloud

In this section, the wing is also used as the fitting geometry model. We are focusing on the running time changed with the number of points in cloud. The number of points is changed from 1731 to 13848. The design parameters and geometry structure are the same except the number of points in cloud. The results of time is listed in Table 5.21.

Table 5.21: Running Time for Wings in Different Number of Points in Cloud

Number of DPs	8							
Number of faces	8							
Number of points	1731	3462	5193	6924	8655	10386	12117	13848
Cycle	10	10	9	8	7	7	7	7
Iterations	131	119	125	116	72	89	107	107
Time/iter	0.69	0.73	0.78	0.78	0.85	0.87	0.89	0.93
Time(optimizer)/iter	0.04	0.08	0.11	0.14	0.18	0.21	0.24	0.27
Total time	98.79	95.43	105.05	97.84	68.71	84.73	102.29	107.39
Total time (optimizer)	10.79	14.56	18.98	21.11	17.96	23.69	30.96	35.10
Per ILM	0.005	0.010	0.013	0.018	0.023	0.027	0.030	0.034
Per Initialization	0.0128	0.0280	0.0405	0.0553	0.0688	0.0843	0.0938	0.1121
Per Derivative	0.0045	0.0077	0.0109	0.0140	0.0184	0.0210	0.0241	0.0274
Per Build	0.41	0.41	0.42	0.40	0.42	0.42	0.41	0.42

As shown in the Figure 5.17, the time per ILM function, the time per initialization/classification and the time for generating sensitives for each design parameter are increased linearly with the number of design parameters because the sparse techniques developed in Chapter 3 reduced the complexity of matrix (and linear equations) calculation into $O(n)$. At the same time, the time per build function is keep constants.

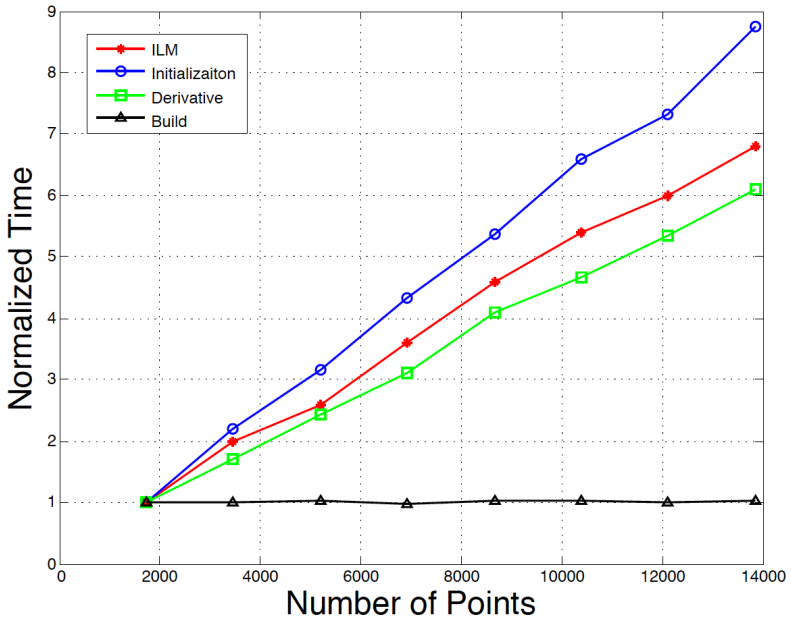


Figure 5.19: Running time analysis based on different number of points

It can be concluded that after applying the algorithm developed in this paper, generating the parametric model from a cloud of points can be solved in the reasonable time.

Chapter 6

Conclusion

6.1 Summary

The objective of this work is to fit a parametric geometry model to a cloud of unclassified points, and to do so accurately and efficiently. The fitting process uses a modified gradient-based optimization method that is applied to the whole cloud, without the need for a user to segment or classify the points. It minimizes the least square of the distances from the geometry model to the points in the cloud.

The basic optimization technique used is the Levenberg-Marquardt algorithm, which is a combination of the gradient descent and Newton's methods. In order to overcome a few shortcomings of the Levenberg-Marquardt algorithm, ideas are borrowed from a heuristic optimization method (Simulated Annealing) to help it improve the fitting accuracy and to help it avoid local minima in the objective function. This new Improved Levenberg-Marquardt (ILM) algorithm also can be used as a good optimization method in other applications, such as traffic equilibrium problems. The results of ILM are compared

with published results, and it shows that the ILM algorithm is generally better than the traditional optimization methods in terms of robustness, efficiency, and accuracy.

In order to apply the fitting algorithm to cases when the initial design parameters are not very good, a new (re)initialization technique and a modified objective function are used. The (re)initialization technique has been designed so as to avoid a problem associated with a poor initial guess, namely the situation when some of the points in the cloud are associated with the discrete points on the opposite side of the configuration. The modified objective function is needed in cases when the initial guess is either too large or too small. This modified objective function reduces the likelihood that the optimizer gets stuck at a local minimum.

Early development of the fitting method was accomplished with fairly simple analytical shapes. For configurations with multiple components, the (re)initialization technique is generalized into a (re)classification technique, in which points in the cloud are associated with a particular component in the configuration, based upon their distance to the various components. To avoid mis-classifications, especially in the early stages of the fitting process, points in the cloud that are near the intersections (as evidenced by the fact that they are equally close to two or more components) are temporarily ignored. As the fitting process proceeds (and the configuration nears the cloud of points), the points near the intersections are added back into the optimization problem. It has been observed that this new method can avoid the mis-classification problem and converges up to 3 cycles faster in the fitting process.

To expand the applicability of this technique beyond simple analytical shapes, the above techniques have been integrated into the Engineering Sketch Pad (`ESP`), which is used to parametrize and define virtually any configuration. Since `ESP` has the ability to compute the sensitivity of the configuration with respect to the design parameters, it has proven

to a be robust and efficient for fairly complex aerospace applications, such as the fitting of a transport-type aircraft (with flow-through engines). The software that couples the fitting process to `ESP` is called `matchCSM`, and it will be made generally available as part of future `ESP` distributions.

The new fitting process has been applied to a variety of configurations. In order to make the algorithm efficient, sparse matrix techniques are employed, which improve the running speed from m^3 to m during the optimization. As a result, the running time for each iteration of the gradient optimization algorithm is shown to increase linearly with the number of points in cloud. This makes the fitting of real-world problems tractable. For example, the plane (transport aircraft with engines) can be fit 21 hours (70 minutes for running optimization algorithm) for 3×10^5 points in the cloud (by `matchCSM` on a MacBook Pro).

6.2 Conclusions

Fitting a parametric geometry model to a cloud of points is accomplished by a new gradient-based optimization method, called the Improved Levenberg-Marquardt (ILM) algorithm. This technique, which combines ideas from the classical Levenberg-Marquardt (LM) method and from the simulated annealing method, helps to improve the fitting accuracy and reduces the likelihood that it finds a local minimum. Because this modification overcomes the local minimum problem, the number of `cycles` (reinitialization/classification) is reduced, thereby improving the overall rate of convergence.

To demonstrate that ILM is a general improvement to LM, it has been applied to a traffic equilibrium problem. By comparing the performance of LM and ILM, it has been shown that ILM generally performs better than LM in terms of accuracy and convergence rate.

A new (re)initialization technique and a modified objective function are key to improving the robustness of the optimization technique. In most cases, this reduces the likelihood that the optimizer will get stuck at a local minimum when the initial guess is not close to the cloud of points. This has been demonstrated by applying the new fitting process to a multiple-component configuration.

Through using the sparse matrix techniques, the running time for each iteration of the gradient-based optimization algorithm is shown to increase linearly with the number of points in cloud. This makes the fitting a large problems tractable.

Using the `ESP` platform, the new algorithm can be applied to many types of parametric models. This whole fitting algorithms is accurate, efficient, and robust when integrated in to `ESP` via the `matchCSM` program.

6.3 Suggested Future Work

There are four suggestions regarding the future works in this research.

Compared to the original LM method, the Improved Levenberg-Marquardt (ILM) algorithm has been shown to improve the possibility that optimization process converges to a global minimum value for the optimization problem. But there is no guarantee that the global minimum value can be reached at the end. For the future work, one can further improve the combination of the gradient-based and heuristic techniques so as to further improve the robustness (possibility of obtaining global minimum value) of the optimization algorithm.

The classification technique introduced here reduce the number of mis-classified points during the fitting process. However, for complex geometry configurations (that contain

many tiny parts), there are still some points that are mis-classified in the intersection areas during the fitting process. For the future, the classification can be further improved for reducing the number of mis-classified points. This will improve the performance of the whole fitting process.

After applying this technique into `ESP`, the fitting process suffers from the long computation times needed for `ESP` to rebuild the configuration with the new set of design parameters; this can be seen by the large `ocsmBuild` time consumed, especially in the "UNION" Boolean operation. For improving the efficiency of the generation of parametric geometry model in the real problem, the "UNION" Boolean operation provided by the `OpenCASCADE` library should be improved.

Currently, the technique has three inputs: a cloud of points, an initial guess of design parameters, and a constructive geometry model. For making the generating parametric model process more automatic and intelligent, one could develop a technique that can determine the constructive geometry model based on the cloud of points automatically. This might use techniques developed for machine vision and learning. Then the inputs of the process can be reduced to two components: a cloud of points and initial guess of design parameters.

Appendix A

Appendix: Pseudocode for MatchCSM

```
// ------------------------------------------------------------------------ //
//                                                                          //
//                                  matchCSM.c                              //
//                            With Classify Technique                       //
//                     with the modified objective function                 //
//    optimization algorithm: Levenberg-Marquardt + Simulated Annealing     //
//                                                                          //
//                           written by Pengcheng Jia                       //
//             Oct.31.2016 at ACML (Aerospace Computational Methods Lab)    //
//                                                                          //
// ------------------------------------------------------------------------ //
```

```
// ------------------------------------------------------------- //
//                                                               //
//                        Declarations                           //
//                    1. sparse Technique                        //
//                    2. min and max value                       //
//                    3. Levenberg-Marquardt                     //
//                                                               //
// ------------------------------------------------------------- //


static  int SPA_SQU(       // A = J'*J
            double Js[],   // Jacobian matrix in sparse format
            double As1[],  // As is the sparse format of A = J' * J As1 is n*n
            double As2[],  // As2 is n * 2*NUM_PITS
            double As3[],  // As3 is 2 * NUM_PITS
            double As4[],  // As4 is NUM_PITS
            int    ntgt,   // number of points
            int    ndv);   // number of design parameters
static  int SPA_MatVec(    // B = J' * g
            double Js[],   // input which is Jacobian matrix sparse format
            double g[],    // input which is gradient vector
            double B[],    // output whihc B is for Ax = B
            int    ntgt,   // number of points
            int    ndv);   // number of design parameters
static  int ApMult(        // Ap = A*p (used in con gradient)
            double As1[],  // n*n
            double As2[],  // n * 2*NUM_PITS
            double As3[],  // 2 * NUM_PITS
```

```
            double As4[],  // NUM_PITS
            double p[],
            double Ap[],   // output of the matrix
            int    ntgt,   // number of points
            int    ndv);   // number of design parameters
static  int ConGrad(       // A*x = b, get -> x
            double As1[],  // input matrix
            double As2[],
            double As3[],
            double As4[],
            double b[],    // input vector
            double x[],    // output vector
            int    ntgt,   // number of points
            int    ndv);   // number of design parameters


static  int Levenberg_Marquart(
            double X[],             // (input)  coordinates of points in cloud
            double Y[],
            double Z[],
            int    dv_ipmtr[],      // (input)  index of design parameters
            int    dv_irow[],       // (input)  index of row
            int    dv_icol[],       // (input)  index of col
            double dv_lbnd[],       // (input)  low bounds for design parameters
            double dv_ubnd[],       // (input)  up bounds for design parameters
            int    ndv,             // (input)  number of design parameters
            int    ntgt,            // (input)  number of points in cloud
            double ParD[],          // (input)  inital design parameters
```

```
            double lambda_ini,      // (input)  initial lambda value
            double cof,             // (input)  cofficient of classification
            double ParD_reslt[],    // output = new design parameters
            int    *iter_reslt,     // output = total iteration number
            double *resi_reslt,     // output = RMS of the distances
            double residual[]);     // output = RMS history



// ------------------------------------------------------------- //
//                                                               //
//                        Main Function                          //
//                                                               //
// ------------------------------------------------------------- //
int main(int        argc,    // number of inputs
         char      *argv[])  // inputs values
{
    ROUTINE(matchCSM); // name the routine
    // if the input format is not correct, return error --------
    if (argc < 5) {
        printf("==> ERROR:: not enough input arguments\n");
        printf("==> The general format of input should be as follow:\n");
        printf("==> csm_model target_points cycles_number classify_coff\n");
        exit(0);
    }

    // load the geometry model ------------------------
    // load process including the intial design parameters in it
```

```
status = ocsmLoad(casefile, &modl);

// build the geometry model -------------------------
buildTo = 0; // for all
nbody   = 0;
status =  ocsmBuild(modl, buildTo, &builtTo, &nbody, NULL);

// get the number of faces for the geometry model --------
for (ibody = 1; ibody <= MODL->nbody; ibody++) {
    if (MODL->body[ibody].onstack != 1) continue;
    Nface = MODL->body[ibody].nface; // number of faces
}

// get the number of design parameters ---------------
ndv = 0;
for (ipmtr = 1; ipmtr <= MODL->npmtr; ipmtr++) {
    if (MODL->pmtr[ipmtr].type == OCSM_EXTERNAL) {
        for (irow = 1; irow <= MODL->pmtr[ipmtr].nrow; irow++) {
            for (icol = 1; icol <= MODL->pmtr[ipmtr].ncol; icol++) {
                ndv++; // accumulated the number of design parameters
            }
        }
    }
}

// get the indexes and bounds of the design parameters --------
ndv = 0;
```

```
for (ipmtr = 1; ipmtr <= MODL->npmtr; ipmtr++) {
    if (MODL->pmtr[ipmtr].type == OCSM_EXTERNAL) {
        int idx = 0;
        for (irow = 1; irow <= MODL->pmtr[ipmtr].nrow; irow++) {
            for (icol = 1; icol <= MODL->pmtr[ipmtr].ncol; icol++) {

                // get the bounds for each design parameter
                status = ocsmGetBnds(modl, ipmtr, irow, icol,
                                     &lbound, &ubound);
                dv_ipmtr[ndv] = ipmtr; // indexs of design parameters
                dv_irow[ ndv] = irow;  // row index
                dv_icol[ ndv] = icol;  // colum index
                // design paramters provided by .csm file
                Par[ ndv] = MODL->pmtr[ipmtr].value[idx];
                dv_lbnd[ndv] = lbound; // up bound of design parameter
                dv_ubnd[ndv] = ubound; // low bound of design parameter

                ndv++;
                idx++;
            }
        }
    }
}

// set the new design parameters based on the bound set in model ------
for (idv = 0; idv < ndv; idv++) {
    // if parameter is lower than low bound
```

```
        if (ParD[idv] < dv_lbnd[idv]){
            ParD[idv] = dv_lbnd[idv];
        }
        // if parameter is bigger than up bound
        if (ParD[idv] > dv_ubnd[idv]){
            ParD[idv] = dv_ubnd[idv];
        }
    }

    // read targt file of cloud of pints ---------------------------
    // count the number of points in cloud
    ntgt = 0;
    fp = fopen(targetfile, "r");
    if (fp == NULL) {
        printf("Error Reading Target File\n");
        exit(0);
    }
    while (1) {
        fscanf(fp, "%lf %lf %lf", &xdum, &ydum, &zdum);
        if (feof(fp)) break;
        ntgt++; // number of points in cloud
    }
    fclose(fp);
    // input the values of target file
    fp = fopen(targetfile, "r");
    for (i = 0; i < ntgt; i++){
        // X,Y,Z are the coordinates of points
```

```
        fscanf(fp, "%lf %lf %lf", &X[i], &Y[i], &Z[i]);
    }
    fclose(fp);

    // loop for reinitial and run Improved Levenberg-Marquardt --------
    status = Levenberg_Marquart(X,Y,Z, dv_ipmtr, dv_irow, dv_icol,
                                dv_lbnd, dv_ubnd, ndv, ntgt, ParD,
                                1, cof, ParD_reslt, &iter_reslt,
                                &resi_reslt, residual);
    iter_total = iter_reslt;    // record the number of iterations
    resi_temp  = resi_reslt;    // temperary residual record
    // record the history of RMS for iterations
    for (i = 0; i < ItMx; i++){
        RMS[i] = residual[i];
    }

    // reinitialization technique ------------------
    for (i = 1; i < CYCLE; i++){
        // reduce the coefficient of classification in each cycle
        cofN-=dCof;
        // be sure the minimum cofN = 1
        if (cofN <= 1){
            cofN = 1;
        }
        // keep the result of design parameters
        for (j = 0; j < ndv; j++){
            ParD[j] = ParD_reslt[j];
```

```
        }
        // run initialize and ILM again
        status = Levenberg_Marquart(X,Y,Z, dv_ipmtr, dv_irow, dv_icol,
                                    dv_lbnd, dv_ubnd, ndv, ntgt, ParD,
                                    1, cofN, ParD_reslt, &iter_reslt,
                                    &resi_reslt, residual);
        // record the RMS history for each cycle
        for (k = 0; k < ItMx; k++){
            RMS[i*ItMx + k] = residual[k];
        }
        // accumulate the number of iterations
        iter_total += iter_reslt;
        CycTol++;
        // break rule
        if (fabs(resi_reslt-resi_temp) < 10e-6){
            printf("==> CYCLE STOP: because the RMS not reduce in 2 cycles\n");
            break;
        }
        resi_temp  = resi_reslt;
    }

    cleanup: // ---------------------------------------------
    FREE(All Variables);
    return status;
}

// ------------------------------------------------------------ //
```

```
//                                                                  //
//                     Matrix multiplication                        //
//                sparse matrix technique applied                   //
//                     (J'*J, J*g, and A*p)                         //
//                                                                  //
// ---------------------------------------------------------------- //


int SPA_SQU(
    double Js[],  // Jacobian matrix in sparse format
    double As1[], // As is the sparse format of A = J' * J As1 is n*n
    double As2[], // As2 is n * 2*NUM_PITS
    double As3[], // As3 is 2 * NUM_PITS
    double As4[], // As4 is NUM_PITS
    int    ntgt,  // number of points
    int    ndv)   // number of design parameters
{
    ROUTINE(SPA_SQU);

    LEN_PARA = ndv;        // number of design parameters
    NUM_PITS = ntgt;       // number of target points
    NUM_PITS3 = 3 * ntgt;  // 3 times number of target points
    LEN_JS    = ndv + 2 ;  // number of cols for Jacobian matrix
    // As1
    for (int i = 0; i < LEN_PARA; i++){
         for (int j = 0; j < LEN_PARA; j++){
            sum = 0.0;
            for (int k = 0; k < NUM_PITS3; k++){
```

```
                sum += Js[k*LEN_JS+i]*Js[k*LEN_JS+j];


            }
            As1[i*LEN_PARA+j] = sum;
        }
    }
    // As2
    for (int i = 0; i < LEN_PARA; i++){
        for (int j = 0; j < NUM_PITS; j++){
            As2[i*2*NUM_PITS+j] = Js[j             *LEN_JS +i] * \
                                  Js[j*             LEN_JS +LEN_PARA]+ \
                                  Js[(j+  NUM_PITS)*LEN_JS +i] * \
                                  Js[(j+  NUM_PITS)*LEN_JS +LEN_PARA]+ \
                                  Js[(j+2*NUM_PITS)*LEN_JS +i] * \
                                  Js[(j+2*NUM_PITS)*LEN_JS +LEN_PARA];
            As2[i*2*NUM_PITS+j+NUM_PITS] \
                                 = Js[j             *LEN_JS +i] * \
                                  Js[j*             LEN_JS +LEN_PARA+1]+ \
                                  Js[(j+  NUM_PITS)*LEN_JS +i] * \
                                  Js[(j+  NUM_PITS)*LEN_JS +LEN_PARA+1]+ \
                                  Js[(j+2*NUM_PITS)*LEN_JS +i] * \
                                  Js[(j+2*NUM_PITS)*LEN_JS +LEN_PARA+1];
        }
    }
    // As3 and As4
    for (int i = 0; i < NUM_PITS; i++){
        As3[i]          = pow(Js[i             *LEN_JS+LEN_PARA],2) + \
```

```
                        pow(Js[(i+  NUM_PITS)*LEN_JS+LEN_PARA],2) + \
                        pow(Js[(i+2*NUM_PITS)*LEN_JS+LEN_PARA],2);
        As3[i+NUM_PITS] = pow(Js[i             *LEN_JS+LEN_PARA+1],2) + \
                        pow(Js[(i+  NUM_PITS)*LEN_JS+LEN_PARA+1],2) + \
                        pow(Js[(i+2*NUM_PITS)*LEN_JS+LEN_PARA+1],2);
        As4[i]          = Js[i             *LEN_JS+LEN_PARA] * \
                          Js[i             *LEN_JS+LEN_PARA+1] + \
                          Js[(i+  NUM_PITS)*LEN_JS+LEN_PARA] * \
                          Js[(i+  NUM_PITS)*LEN_JS+LEN_PARA+1] + \
                          Js[(i+2*NUM_PITS)*LEN_JS+LEN_PARA] * \
                          Js[(i+2*NUM_PITS)*LEN_JS+LEN_PARA+1];
    }
    return status;
}

int SPA_MatVec(
    double Js[],  // input which is Jacobian matrix sparse format
    double g[],   // input which is gradient vector
    double B[],   // output whihc B is for Ax = B
    int    ntgt,  // number of points
    int    ndv)   // number of design parameters

{
    ROUTINE(SPA_MatVec);
    // the elements B(1:n)
    for (int i = 0; i < LEN_PARA; i++){
        sum = 0.0;
```

```
        for (int j = 0; j < NUM_PITS3; j++){
            sum += Js[j*LEN_JS + i] * g[j];
        }
        B[i] = sum;
    }
    // the elements B(1:n)
    for (int i = 0; i < NUM_PITS; i++){ ?
        B[LEN_PARA+i] =          Js[ i            *LEN_JS + LEN_PARA]   * \
                                 g[           i] + \
                                 Js[(i+  NUM_PITS)*LEN_JS + LEN_PARA]   * \
                                 g[  NUM_PITS+i] + \
                                 Js[(i+2*NUM_PITS)*LEN_JS + LEN_PARA]   * \
                                 g[2*NUM_PITS+i];
        B[LEN_PARA+i+NUM_PITS] = Js[ i            *LEN_JS + LEN_PARA+1] * \
                                 g[           i] + \
                                 Js[(i+  NUM_PITS)*LEN_JS + LEN_PARA+1] * \
                                 g[  NUM_PITS+i] + \
                                 Js[(i+2*NUM_PITS)*LEN_JS + LEN_PARA+1] * \
                                 g[2*NUM_PITS+i];
    }
    return status;
}

int ApMult(
    double As1[], // n*n
    double As2[], //
    double As3[],
```

```
    double As4[],
    double p[],
    double Ap[],  // output of the matrix
    int    ntgt,  // number of points
    int    ndv)   // number of design parameters
{
    ROUTINE(ApMult);
    // Ap(1:n)
    for (int i = 0; i < LEN_PARA; i++){
        sum1 = 0.0;
        sum2 = 0.0;
        for (int j = 0; j < NUM_PARA2; j++){
            if (j < LEN_PARA){
                sum1 += As1[i*LEN_PARA+j] * p[j];
            }else if (j >= LEN_PARA){
                sum2 += As2[i*LEN_COOR + j-LEN_PARA] * p[j];
            }
        }
        Ap[i] = sum1 + sum2;
    }
    // Ap(n+1:2*m+n)
    for (int i = 0; i < NUM_PITS; i++){
        sum3 = 0.0;
        sum4 = 0.0;
        for (int j = 0; j < LEN_PARA; j++){
            sum3 += As2[j*LEN_COOR + i]         * p[j];
            sum4 += As2[j*LEN_COOR + i+NUM_PITS]* p[j];
```

```
        }
        Ap[LEN_PARA + i           ] = sum3 + As3[i]         *\
                                      p[LEN_PARA + i]          + \
                                      As4[i]*p[LEN_PARA+NUM_PITS+i];
        Ap[LEN_PARA + i + NUM_PITS] = sum4 + As3[i+NUM_PITS]*\
                                      p[LEN_PARA + NUM_PITS+i] + \
                                      As4[i]*p[LEN_PARA+i];
    }
    return status;
}

// ------------------------------------------------------------- //
//                                                               //
//   Solving Systems of Linear Equations by LU decompostion      //
//                                                               //
// ------------------------------------------------------------- //
int ConGrad(       // A*x = b, get -> x
    double As1[],  // input matrix
    double As2[],
    double As3[],
    double As4[],
    double b[],    // input vector
    double x[],    // output vector
    int    ntgt,   // number of points
    int    ndv)    // number of design parameters
{
    double ERR = 1e-10;
```

```
ROUTINE(ConGrad);
for (int i = 0; i < NUM_PARA2; i++){
    r[i] = b[i]; // r = b - A*x, because initial x = 0, r = b
    p[i] = r[i]; // initial p = r,
}
double rs_old = 0.0;
for (int i = 0; i < NUM_PARA2; i++){ // rs_old = r'*r
    rs_old += pow(r[i],2);
}
if (fabs(rs_old -0.0) < 1e-10){
    for (int i = 0; i < NUM_PARA2; i++){
        x[i] = 0.0;
    }
}
int IterMax = 50;
double beta = 0.0;
double rs_new = 0.0;
// if residual is not equal 0
if (!(fabs(rs_old - 0.0) < 1e-10)){
    // iteration loop
    for (int iter = 0; iter < IterMax; iter++){
        // Ap = A*p
        ApMult(As1,As2,As3,As4,p,Ap,ntgt,ndv);
        // beta is a scalar
        beta = 0.0;
        for (int i = 0; i < NUM_PARA2; i++){
            beta += p[i]*Ap[i];
```

```
            }
            alpha = rs_old/beta; //alpha is a scalar
            // get new r value and x value
            for (int i = 0; i < NUM_PARA2; i++){
                x[i] = x[i]+ alpha* p[i];
                r[i] = r[i]- alpha*Ap[i];
            }
            // rs_new = r'*r is a scalar
            rs_new = 0.0;
            for (int i = 0; i < NUM_PARA2; i++){
                rs_new += pow(r[i],2);
            }
            // stopping rule
            if (sqrt(rs_new)<ERR){
                rs_old = rs_new;
                break;
            }
            // update p value which is vector
            for (int i = 0; i < NUM_PARA2; i++){
                p[i] = r[i] + (rs_new/rs_old)*p[i];
            }
            // update the rs_old value
            rs_old = rs_new;
        }
    }

    cleanup: // ----------------------------
```

```
    FREE(ALL VARIABLES);
    return status;
}

// ------------------------------------------------------------- //
//                                                               //
//           Improved Levenberg-Marquardt Technique (ILM)        //
//          include the initialization, classification, LM       //
//                                                               //
// -------------------------------------------------------------//

int Levenberg_Marquart(
    double X[],             // (input)  coordinates of points in cloud
    double Y[],
    double Z[],
    int    dv_ipmtr[],      // (input)  index of design parameters
    int    dv_irow[],       // (input)  index of row
    int    dv_icol[],       // (input)  index of col
    double dv_lbnd[],       // (input)  low bounds for design parameters
    double dv_ubnd[],       // (input)  up bounds for design parameters
    int    ndv,             // (input)  number of design parameters
    int    ntgt,            // (input)  number of points in cloud
    double ParD[],          // (input)  inital design parameters
    double lambda_ini,      // (input)  initial lambda value
    double cof,             // (input)  cofficient of classification
    double ParD_reslt[],    // output = new design parameters
    int    *iter_reslt,     // output = total iteration number
```

```
    double *resi_reslt,     // output = RMS of the distances
    double residual[])      // output = RMS history
{
    DEFINE(ALL VARIABLES)
    ROUTINE(Levenberg_Marquart);

    // build the new model based on the input design parameters ------
    for (idv = 0; idv < ndv; idv++) {
        status = ocsmSetValuD(modl, dv_ipmtr[idv],
                              dv_irow[idv], dv_icol[idv], ParD[idv]);
    }
    buildTo = 0; // for all
    nbody   = 0;
    status =  ocsmBuild(modl, buildTo, &builtTo, &nbody, NULL);

    // generate discrete points on the geometry model ----------------------
    // get the number of faces on the body which is on the stack
    for (ibody = 1; ibody <= MODL->nbody; ibody++) {
        if (MODL->body[ibody].onstack != 1) continue;
        Nface = MODL->body[ibody].nface;
    }
    // get the total number of discrete points in model
    for (ibody = 1; ibody <= MODL->nbody; ibody++) {
        if (MODL->body[ibody].onstack != 1) continue;
        // get the bound of geometry
        status = EG_getBoundingBox(MODL->body[ibody].ebody, box);
        // get whole size of geometry
```

```
size   = sqrt(SQR(box[3]-box[0]) + \
             SQR(box[4]-box[1]) + \
             SQR(box[5]-box[2]));
// set the u,v space
params[0] = 0.0100 * size;
params[1] = 0.0050 * size;
params[2] = 15.0;
// generate the discrete pints on body which is on stack
status = EG_makeTessBody(MODL->body[ibody].ebody, params,
                         &(MODL->body[ibody].etess));
// get the xyz and uv value from the geometry model
ndpnt = 0; // initial the total of discrete pints is 0
PintsIdx[0] = ndpnt; // start point index of each face
for (iface = 1; iface <= MODL->body[ibody].nface; iface++){
    status = EG_getTessFace(MODL->body[ibody].etess, iface,
                            &npnt, &xyz, &uv, &ptype, &pindx,
                            &ntri, &tris, &tric);
    // this loop for get the number of discete pints on model
    ndpnt +=  npnt;
    Pints[iface-1] = npnt;   // number of points in each face
    PintsIdx[iface] = ndpnt; // start point index of each face
    // get the attribute of each face
    status =  EG_attributeRet(MODL->body[ibody].face[iface].eface,
                            "_faceID", &atype, &len, &ints,
                            &reals, &str);
    for (ipnt = 0; ipnt < npnt; ipnt++){
        // record the coordinates of discrete points
```

```
                xd[ndpnt+ipnt] = xyz[ipnt*3];
                yd[ndpnt+ipnt] = xyz[ipnt*3+1];
                zd[ndpnt+ipnt] = xyz[ipnt*3+2];
                ud[ndpnt+ipnt] = uv[ipnt*2];
                vd[ndpnt+ipnt] = uv[ipnt*2+1];
                // record the body and face id
                dbody[ndpnt+ipnt] = ibody;
                dface[ndpnt+ipnt] = iface;
                // record the attribute of each face
                fID1d[ndpnt+ipnt] = ints[0];
                fID2d[ndpnt+ipnt] = ints[1];
                fID3d[ndpnt+ipnt] = ints[2];
            }
        }
    }

    // classification and initialization process ------------------------
    // initial the number of points will be used in fitting process
    int     Cnpnt = 0;
    // if the cof > 1, less points in cloud are counted to as fitting target
    if (cof > 1){
        Cnpnt = 0;
        // find the shortest distances from discrete points for each target
        for (i = 0; i < ntgt; i++){
            // for each point in cloud, loop the distrete points
            for (k = 0; k < ndpnt; k++){
                dis[k] = pow(X[i] - xd[k],2) + \
```

```
                pow(Y[i] - yd[k],2) + \
                pow(Z[i] - zd[k],2);
    }
    // get the array of shortest distance
    // from one target pints to each face
    for (iface = 0; iface < Nface; iface++){
        double valDis   = dis[PintsIdx[iface]];
        int    indexDis = PintsIdx[iface];
        // loop all discrete points on each face
        // find the shortest distance for each face
        for (ipnt = PintsIdx[iface]; ipnt < PintsIdx[iface+1]; ipnt++){
            if (valDis > dis[ipnt]){
                valDis = dis[ipnt];
                indexDis = ipnt;
            }
        }
        // disMins is the minimum distance for each face
        // disMinsIdx is the index of minimum distance for each face
        disMins[iface]     = valDis;
        disMinsIdx[iface]  = indexDis;
    }
    // the minimum distance * cofficient is still the minmum distance,
    // then record this discrete point
    double disMinF      = min2(disMins,Nface);
    int    disMinFIdx   = min (disMins,Nface);
    // use (minimum distance * cofficient) replace minimum distance
    double disMinF_iter = cof*disMinF;
```

```
        disMins[disMinFIdx] = disMinF_iter;
        // if it is still the shortest distance,
        // record all information of this discrete point
        if (fabs(disMinF_iter - min2(disMins,Nface)) < 10e-16){
            // record the related discrete points
            Xd[Cnpnt]  = xd[disMinsIdx[disMinFIdx]];
            Yd[Cnpnt]  = yd[disMinsIdx[disMinFIdx]];
            Zd[Cnpnt]  = zd[disMinsIdx[disMinFIdx]];
            Ud[Cnpnt]  = ud[disMinsIdx[disMinFIdx]];
            Vd[Cnpnt]  = vd[disMinsIdx[disMinFIdx]];
            ibd[Cnpnt] = dbody[disMinsIdx[disMinFIdx]];
            ifd[Cnpnt] = dface[disMinsIdx[disMinFIdx]];
            fID1[Cnpnt]= fID1d[disMinsIdx[disMinFIdx]];
            fID2[Cnpnt]= fID2d[disMinsIdx[disMinFIdx]];
            fID3[Cnpnt]= fID3d[disMinsIdx[disMinFIdx]];
            // record the related target points
            Xc[Cnpnt]  = X[i];
            Yc[Cnpnt]  = Y[i];
            Zc[Cnpnt]  = Z[i];
            Cnpnt++;
        }
    }
    // find the shortest distances from cloud points for each discrete
    for (i = 0; i < ndpnt; i++){
        // for each discrete point, loop the cloud points
        for (k = 0; k < ntgt; k++){
            dis_DtP[k] = pow(X[k] - xd[i],2) + \
```

```
                        pow(Y[k] - yd[i],2) + \
                        pow(Z[k] - zd[i],2);
            }
            // in the target points,
            // record the index of the minimum distance for discrete
            idx_dis[i] = min(dis_DtP,ntgt);
            // record the related discrete points
            Xd[Cnpnt]  = xd[i];
            Yd[Cnpnt]  = yd[i];
            Zd[Cnpnt]  = zd[i];
            Ud[Cnpnt]  = ud[i];
            Vd[Cnpnt]  = vd[i];
            ibd[Cnpnt] = dbody[i];
            ifd[Cnpnt] = dface[i];
            fID1[Cnpnt]= fID1d[i];
            fID2[Cnpnt]= fID2d[i];
            fID3[Cnpnt]= fID3d[i];
            // record the related target points
            Xc[Cnpnt]  = X[idx_dis[i]];
            Yc[Cnpnt]  = Y[idx_dis[i]];
            Zc[Cnpnt]  = Z[idx_dis[i]];
            // increase the number of total points will be fitted
            Cnpnt++;
        }
    }
    // if the cof = 1, the total number of points = target + discrete
    // almost the same process as (cof > 1)
```

```
if (cof == 1){
    Cnpnt = ntgt+ndpnt;
    // find the shortest distances from discrete points for each target
    for (i = 0; i < ntgt; i++){
        for (k = 0; k < ndpnt; k++){
            dis[k] = pow(X[i] - xd[k],2) + \
                     pow(Y[i] - yd[k],2) + \
                     pow(Z[i] - zd[k],2);
        }
        idx_dis[i] = min(dis,ndpnt);
        // record the related discrete points
        Xd[i]  = xd[idx_dis[i]];
        Yd[i]  = yd[idx_dis[i]];
        Zd[i]  = zd[idx_dis[i]];
        Ud[i]  = ud[idx_dis[i]];
        Vd[i]  = vd[idx_dis[i]];
        ibd[i] = dbody[idx_dis[i]];
        ifd[i] = dface[idx_dis[i]];
        fID1[i]= fID1d[idx_dis[i]];
        fID2[i]= fID2d[idx_dis[i]];
        fID3[i]= fID3d[idx_dis[i]];
        // record the related target points
        Xc[i]  = X[i];
        Yc[i]  = Y[i];
        Zc[i]  = Z[i];
    }
    // find the shortest distances from cloud points for each discrete
```

```
        for (i = 0; i < ndpnt; i++){
            // for each discrete point, loop the target points
            for (k = 0; k < ntgt; k++){
                dis_DtP[k] = pow(X[k] - xd[i],2) + \
                             pow(Y[k] - yd[i],2) + \
                             pow(Z[k] - zd[i],2);
            }
            // in the target points
            // record the index of the minimum distance for point i
            idx_dis[i] = min(dis_DtP,ntgt);
            // record the related discrete points
            Xd[i+ntgt]  = xd[i];
            Yd[i+ntgt]  = yd[i];
            Zd[i+ntgt]  = zd[i];
            Ud[i+ntgt]  = ud[i];
            Vd[i+ntgt]  = vd[i];
            ibd[i+ntgt] = dbody[i];
            ifd[i+ntgt] = dface[i];
            fID1[i+ntgt]= fID1d[i];
            fID2[i+ntgt]= fID2d[i];
            fID3[i+ntgt]= fID3d[i];
            // record the related target points
            Xc[i+ntgt]  = X[idx_dis[i]];
            Yc[i+ntgt]  = Y[idx_dis[i]];
            Zc[i+ntgt]  = Z[idx_dis[i]];
        }
    }
```

```
// define the parameters which will be used later
NUM_PARA2 = ndv+2*Cnpnt;  // the length of design + parametric coordinates
LEN_PARA  = ndv;          // the number of design parameters
LEN_COOR  = 2*Cnpnt;      // number of total parametric coordinates (u,v)
NUM_PITS  = Cnpnt;        // number of one parametric coordinates u
NUM_PITS3 = 3 * Cnpnt;    // 3 times number of target points
LEN_JS    = ndv + 2 ;     // number of cols for Jacobian matrix
mm = LEN_JS*NUM_PITS3;    // number of elements in Jacobian matrix
epsilon = 10e-10;         // tolerance (small threshold)
IterMax = 30;             // maximal number of (main) iterations
lambda_sqrt = sqrt(lambda_ini);  // the lambda value

// generate objective function --------------------------
for (i = 0; i < NUM_PITS; i++){
        gx[i] = (-Xc[i]+Xd[i]);
        gy[i] = (-Yc[i]+Yd[i]);
        gz[i] = (-Zc[i]+Zd[i]);
}
// combine the gx,gy,gz together
for (i = 0; i < NUM_PITS; i++){
    g[i           ] = gx[i];
    g[i+  NUM_PITS] = gy[i];
    g[i+2*NUM_PITS] = gz[i];
}

// generate the deritaive of the objective function ---------------
for (i = 0; i < NUM_PITS; i++){
```

```
        resi += (pow(gx[i],2)+pow(gy[i],2)+pow(gz[i],2));
    }
    resi = sqrt(resi/(NUM_PITS3));      // initial residual
    residual[0] = resi;                 // record this into array

    // generate the derivative of design parameters --------------------
    // generate the derivatives for each design parameter
    for (idv = 0; idv < ndv; idv++) {
        // pick which design parameter will be derivatived
        status = ocsmSetVelD(modl, 0,     0,    0,    0.0);
        status = ocsmSetVelD(modl, dv_ipmtr[idv], dv_irow[idv],
                             dv_icol[idv], 1.0);
        // build the model
        buildTo = 0;
        nbody   = 0;
        status =  ocsmBuild(modl, buildTo, &builtTo, &nbody, NULL);
        // generate the derivative of design parameters
        for (i = 1; i <= Nface; i++){
            int uvid = 0;
            // generate the UVs will be used in ocsmGetVel
            for (j = NumFacIdx[i-1]; j < NumFacIdx[i]; j++){
                UVs[uvid*2+0] = Us[j];
                UVs[uvid*2+1] = Vs[j];
                uvid++;
            }
            // get the sensitivity for each point
            status =  ocsmGetVel(modl, ibd[1], OCSM_FACE,
```

```
                                i, NumFac[i], UVs, Vels);
            // put the sensitivity into Jacobian
            int velid = 0;
            for (j = NumFacIdx[i-1]; j < NumFacIdx[i]; j++){
                // derivative of design parameter for x
                J[ Pid[j]            *LEN_JS + idv] = Vels[velid*3+0];
                // derivative of design parameter for y
                J[(Pid[j]+  NUM_PITS)*LEN_JS + idv] = Vels[velid*3+1];
                // derivative of design parameter for z
                J[(Pid[j]+2*NUM_PITS)*LEN_JS + idv] = Vels[velid*3+2];
                velid++;
            }
        }
    }

    // generate the derivative of parametric coordinates ----------------
    for (itgt = 0; itgt < NUM_PITS; itgt++) {
        uv_der[0] = Ud[itgt];
        uv_der[1] = Vd[itgt];
        // generate the derivative of u and v
        status =  EG_evaluate(MODL->body[ibd[itgt]].face[ifd[itgt]].eface,
                              uv_der, eval);
        // write that into Jacobian
        // derivative of u
        J[itgt             *LEN_JS + LEN_PARA]   = eval[3];
        J[(itgt+  NUM_PITS)*LEN_JS + LEN_PARA]   = eval[4];
        J[(itgt+2*NUM_PITS)*LEN_JS + LEN_PARA]   = eval[5];
```

```
        // derivative of v
        J[itgt             *LEN_JS + LEN_PARA +1]   = eval[6];
        J[(itgt+  NUM_PITS)*LEN_JS + LEN_PARA +1]   = eval[7];
        J[(itgt+2*NUM_PITS)*LEN_JS + LEN_PARA +1]   = eval[8];
    }


    // Improved Levenberg-Marquardt method ----------------------------
    // par is ParD + u + v
    for (int i = 0; i < LEN_PARA; i++){
        par[i] = ParD[i];
    }
    for (int i = 0; i < NUM_PITS; i++){
        par[i+LEN_PARA         ] = Ud[i];
        par[i+LEN_PARA+NUM_PITS] = Vd[i];
    }
    // initial the ifd new and resi_temp
    for (i = 0; i < NUM_PITS; i++){
         ifd_new[i] = ifd[i];
    }
    resi_temp = resi;
    // begin of the ILM loop --------
    for (iter = 1; iter <= IterMax; iter++){
        // remove the panelty term after 5 iterations
        if (iter == 6){
            NUM_PITS  = Cnpnt-ndpnt;
            NUM_PARA2 = ndv+2*NUM_PITS;
            LEN_PARA  = ndv;
```

```
LEN_COOR  = 2*NUM_PITS;
NUM_PITS3 = 3 * NUM_PITS;
LEN_JS    = ndv + 2 ;
mm = LEN_JS*NUM_PITS3;
// rewrite the Jacobian
for (itgt = 0 ; itgt < NUM_PITS; itgt++){
    for (idv = 0; idv < LEN_PARA; idv++){
        // for design parameter
        J[itgt             *LEN_JS + idv] = \
        J[itgt          *LEN_JS + idv];
        J[(itgt+  NUM_PITS)*LEN_JS + idv] = \
        J[(itgt+  Cnpnt)*LEN_JS + idv];
        J[(itgt+2*NUM_PITS)*LEN_JS + idv] = \
        J[(itgt+2*Cnpnt)*LEN_JS + idv];
        // for u
        J[itgt             *LEN_JS + LEN_PARA]   = \
        J[itgt          *LEN_JS + LEN_PARA];
        J[(itgt+  NUM_PITS)*LEN_JS + LEN_PARA]   = \
        J[(itgt+  Cnpnt)*LEN_JS + LEN_PARA];
        J[(itgt+2*NUM_PITS)*LEN_JS + LEN_PARA]   = \
        J[(itgt+2*Cnpnt)*LEN_JS + LEN_PARA];
        // for v
        J[itgt             *LEN_JS + LEN_PARA +1]   = \
        J[itgt          *LEN_JS + LEN_PARA +1];
        J[(itgt+  NUM_PITS)*LEN_JS + LEN_PARA +1]   = \
        J[(itgt+  Cnpnt)*LEN_JS + LEN_PARA +1];
        J[(itgt+2*NUM_PITS)*LEN_JS + LEN_PARA +1]   = \
```

```
                J[(itgt+2*Cnpnt)*LEN_JS + LEN_PARA +1];
            }
        }
        // rewrite objective function
        for (i = 0; i < NUM_PITS; i++){
            g[i           ] = g[i        ];
            g[i+  NUM_PITS] = g[i+  Cnpnt];
            g[i+2*NUM_PITS] = g[i+2*Cnpnt];
        }
        // rewrite par
        for (i = 0; i < NUM_PITS; i++){
            par[i+LEN_PARA         ] = par[i+LEN_PARA      ];
            par[i+LEN_PARA+NUM_PITS] = par[i+LEN_PARA+Cnpnt];
        }
    }
    // mutiply the J'*J = A (A is formed by As1,As2,As3,As4)
    SPA_SQU(J,As1,As2,As3,As4,NUM_PITS,LEN_PARA);
    // A + unit matrix
    for (i = 0; i < LEN_PARA; i++){
        As1[i*LEN_PARA+i] += lambda_sqrt;
    }
    for (i = 0; i < LEN_COOR; i++){
        As3[i] += lambda_sqrt;
    }
    // output -> B = J'*g
    SPA_MatVec(J,g,B,NUM_PITS,LEN_PARA);
    // Conjointed gradient calculate the x of Ax = B
```

```
// get the del_par which is the delta of par
ConGrad(As1,As2,As3,As4,B,del_par,NUM_PITS,LEN_PARA);
progress = 0.0;  // initial progress = 0
for (i = 0; i < NUM_PARA2; i++){
    // norm of the progress
    progress += pow(del_par[i],2);
}
// stopping rule
if (progress < epsilon){
    break;
}
// iteration fomular
for (i = 0; i < NUM_PARA2; i++){
    par_temp[i] = par[i] - del_par[i];
}
// fix the bound of design parameters
for (idv = 0; idv < LEN_PARA; idv++) {
    // if parameter is lower than low bound
    if (par_temp[idv] < dv_lbnd[idv]){
        par_temp[idv] = dv_lbnd[idv];
    }
    // if parameter is bigger than up bound
    if (par_temp[idv] > dv_ubnd[idv]){
        par_temp[idv] = dv_ubnd[idv];
    }
}
// seperate the par into dv and uv
```

```
for (i = 0; i < LEN_PARA; i++){
    ParD_temp[i] = par_temp[i];
}
for (i = 0; i < NUM_PITS; i++){
    Ud_temp[i] = par_temp[i+LEN_PARA];
    Vd_temp[i] = par_temp[i+LEN_PARA+NUM_PITS];
}

// for regenerate the geometry model -------------------------
// set the new design parameters into model
for (idv = 0; idv < ndv; idv++) {
    status = ocsmSetValuD(modl, dv_ipmtr[idv],
                          dv_irow[idv], dv_icol[idv], ParD_temp[idv]);
}
// build model
buildTo = 0;
nbody   = 0;
status =  ocsmBuild(modl, buildTo, &builtTo, &nbody, NULL);

// if the numbers of faces are same----------------
if (MODL->body[ibd[1]].nface == nfaceOld){
    // creat the new face's atrribute (translate table)
    for (ibody = 1; ibody <= MODL->nbody; ibody++) {
        if (MODL->body[ibody].onstack != 1) continue;
        for (iface = 1; iface <= MODL->body[ibody].nface; iface++){
            // get the attribute of each face
            EG_attributeRet(MODL->body[ibody].face[iface].eface,
```

```
                        "_faceID", &atype, &len, &ints,
                        &reals, &str);
            // record the face id
            fID1_inter[iface-1] = ints[0];
            fID2_inter[iface-1] = ints[1];
            fID3_inter[iface-1] = ints[2];
            ifd_inter[iface-1]= iface;
        }
    }
    // translate the face id based on the new model
    for (i = 0; i < NUM_PITS; i++){
        for (j = 0; j < nfaceOld; j++){
            if (fID1[i] == fID1_inter[j] && \
                fID2[i] == fID2_inter[j] && \
                fID3[i] == fID3_inter[j]){
                ifd_new[i]  = ifd_inter[j];
                fID1_new[i] = fID1_inter[j];
                fID2_new[i] = fID2_inter[j];
                fID3_new[i] = fID3_inter[j];
            }
        }
    }
    // Generate the discrete points based on the new UV
    for (i = 0; i < NUM_PITS; i++){
        uv_disc[0] = Ud_temp[i];
        uv_disc[1] = Vd_temp[i];
        status = ocsmGetXYZ(modl, ibd[i], OCSM_FACE,
```

```
                                ifd_new[i], 1, uv_disc, xyz_disc);
            Xd_temp[i] = xyz_disc[0];
            Yd_temp[i] = xyz_disc[1];
            Zd_temp[i] = xyz_disc[2];
        }

    // if the numbers of faces are different ----------------
    // update the design parameters and break
    }else if (MODL->body[ibody].nface != nfaceOld){
        // updata par
        for (int i = 0; i < NUM_PARA2; i++){
            par[i] = par_temp[i];
        }
        break;
    }
    // updated the result of the face attribute
    for (int i = 0; i < NUM_PITS; i++){
        fID1[i] = fID1_new[i];
        fID2[i] = fID2_new[i];
        fID3[i] = fID3_new[i];
    }

    // regenerate the objective function ----------------------
    for (i = 0; i < NUM_PITS; i++){
            gx_temp[i] = (-Xc[i]+Xd_temp[i]);
            gy_temp[i] = (-Yc[i]+Yd_temp[i]);
            gz_temp[i] = (-Zc[i]+Zd_temp[i]);
```

```
}
for (i = 0; i < NUM_PITS; i++){
    g_temp[i]            = gx_temp[i];
    g_temp[i+  NUM_PITS] = gy_temp[i];
    g_temp[i+2*NUM_PITS] = gz_temp[i];
}

// norm of the objective function ---------------------------
resi_temp = 0.0;
for (i = 0; i < NUM_PITS; i++){
    resi_temp += (pow(gx_temp[i],2)+\
                  pow(gy_temp[i],2)+\
                  pow(gz_temp[i],2));
}
resi_temp = sqrt(resi_temp/(NUM_PITS3));
residual[iter] = resi_temp;

// recalculate the Jacobian matrix --------------------------
// generate the derivative of design parameters
for (idv = 0; idv < LEN_PARA; idv++){
    // pick which design parameter will be derivatived
    status = ocsmSetVelD(modl, 0,     0,    0,    0.0);
    status = ocsmSetVelD(modl, dv_ipmtr[idv], dv_irow[idv],
                         dv_icol[idv], 1.0);
    // build the model
    buildTo = 0;
    nbody   = 0;
```

```
        status =  ocsmBuild(modl, buildTo, &builtTo, &nbody, NULL);
        // generate the derivative of design parameters
        for (i = 1; i <= Nface; i++){
            // generate the UVs will be used in ocsmGetVel
            int uvid = 0;
            for (j = NumFacIdx[i-1]; j < NumFacIdx[i]; j++){
                UVs[uvid*2+0] = Us[j];
                UVs[uvid*2+1] = Vs[j];
                uvid++;
            }
            // get the sensitivity for each point
            status =  ocsmGetVel(modl, ibd[1], OCSM_FACE, i,
                                 NumFac[i], UVs, Vels);
            // put the sensitivity into Jacobian
            int velid = 0;
            for (j = NumFacIdx[i-1]; j < NumFacIdx[i]; j++){
                // derivative of design parameter for coordinates
                J_temp[ Pid[j]            *LEN_JS + idv] = Vels[velid*3+0];
                J_temp[(Pid[j]+  NUM_PITS)*LEN_JS + idv] = Vels[velid*3+1];
                J_temp[(Pid[j]+2*NUM_PITS)*LEN_JS + idv] = Vels[velid*3+2];
                velid++;
            }
        }
    }

    // generate the derivative of u and v ----------------
    for (itgt = 0; itgt < NUM_PITS; itgt++) {
```

```
        uv_der[0] = Ud_temp[itgt];
        uv_der[1] = Vd_temp[itgt];
        // generate the derivative of u and v
        EG_evaluate(MODL->body[ibd[itgt]].face[ifd_new[itgt]].eface,
                    uv_der, eval);
        // derivative of u
        J_temp[itgt             *LEN_JS + LEN_PARA]   = eval[3];
        J_temp[(itgt+  NUM_PITS)*LEN_JS + LEN_PARA]   = eval[4];
        J_temp[(itgt+2*NUM_PITS)*LEN_JS + LEN_PARA]   = eval[5];
        // derivative of v for z
        J_temp[itgt             *LEN_JS + LEN_PARA +1]   = eval[6];
        J_temp[(itgt+  NUM_PITS)*LEN_JS + LEN_PARA +1]   = eval[7];
        J_temp[(itgt+2*NUM_PITS)*LEN_JS + LEN_PARA +1]   = eval[8];
    }

    // ILM method update parameter rule ----------------------
    // accept, improvement
    if (residual[iter] < residual[iter-1]){
        // reduce lambda, move to next iteration
        lambda_sqrt = lambda_sqrt/2;
        // updata par
        for (int i = 0; i < NUM_PARA2; i++){
            par[i] = par_temp[i];
        }
        // updata g which is the "gradient vector" of objective
        for (int i = 0; i < NUM_PITS3; i++){
            g[i] = g_temp[i];
```

```
        }
        // updata J which is the Jacobian of objective
        for (int i = 0; i < mm; i++){
            J[i] = J_temp[i];
        }
    // not accept
    }else{
        // increase lambda, recompute the step
        lambda_sqrt = lambda_sqrt*2;
        // Simulated Annealing which accept result if increaese a little
        if (residual[iter] < exp(progress)*residual[iter-1]){
            for (int i = 0; i < NUM_PARA2; i++){
                par[i] = par_temp[i]; // updata par
            }
            for (int i = 0; i < NUM_PITS3; i++){
                g[i] = g_temp[i]; // updata par
            }
            for (int i = 0; i < mm; i++){
                J[i]  = J_temp[i]; // updata J
            }
        }else{
            residual[iter] = residual[iter-1];
            resi_temp = residual[iter];
            if (fabs(residual[iter] - residual[iter-4]) < 10e-8){
                break;
            }
        }
```

```
        }
    }
    // output the result design parameters
    for (i = 0; i < LEN_PARA; i++){
        ParD_reslt[i] = par[i];
    }
    // output the iteration number and final RMS
    *iter_reslt = iter;
    *resi_reslt = resi_temp;

    cleanup: // ----------------------------
    FREE(ALL VARIABLES);
    return status;
}
```

Appendix B

Appendix: Readme File for Using MatchCSM

```
*****************************************************************
*                                                               *
*                   ReadMe file for matchCSM                    *
*       This is the program for fitting the parametric model    *
*                   from the cloud of points                    *
*                                                               *
*                    Written by Pengcheng Jia                   *
*                      Nov.09.2016 at ACML                      *
*****************************************************************


This folder contains 3 files and one folder.
They are Makefile, matchCSM.c, GenPintsCSM.c and Demo folder.
```

```
You should download the Engineering Sketch Pad (ESP) as the platform
and OpenCASCAD as geometry library first.

The web you can download these is as following:
http://raphael.mit.edu/ESP/   for OCC680 (choose based on your system)
http://raphael.mit.edu/ESP/archive/ for ESP1.09.tgz


1.================ How to compile/build the code ================

1.1 cd ~/$DISROOT                set the ESP folder as root folder
1.2 Copy the "Makefile" into the $DISROOT/src/OpenCSM. Replace the
the original one.
1.3 Copy the "matchCSM.c and GenPintsCSM.c " into the $DISROOT/src/OpenCSM
1.4 Follow the same instruction of building ESP

(2. easy run for testing, if you choose this step, please ignore step 2)
2.1 get into ESPMatchCSM/Demo/Fuselage6
   2.2 Copy the "fuselage6Pints.txt and fuselage6Ini.csm "
   into the $DISROOT/bin
   2.3 type the follow into terminal:
   cd ~/$DISROOT/bin ./matchCSM fuselage6Ini fuselage6Pints 15 1.2


2.================== How to use the program ==================
```

```
2.1 "GenPintsCSM.c" (Generate the points of cloud), If you already have
the points data that need to be fitted, then skip this process
2.1.1 Open the terminal window
2.1.2 Type the following command in the terminal:
  cd ~/$DISROOT/bin ./GenPintsCSM GeometryName(.csm)
  GeometryName(.csm) is the input of GenPintsCSM function.
  It is the geometry model that contains the correct design
  parameters. Some examples of GeometryName.csm can be
  found in Demo folder.
  Example: ESPMatchCSM/Demo/Box/box.csm
2.1.3 Output results
  After running this process, the cloud of points
  file will be generated in the bin folder and named as
  1. "GeometryNamePints"  :XYZ of the each points in cloud
  2. "GeometryNamePints2" :XYZ and face number of each points in cloud
  3. "GeometryNamePar"    :design parameters for generating the points

2.2 "matchCSM.c" (Run the fitting algorithm for the points of cloud)
2.2.1 Open the terminal window
2.2.2 Type the following command in the terminal:
  cd ~/$DISROOT/bin ./matchCSM GeometryNameIni(.csm)
  GeometryNamePints(.txt) Cycle Coefficient
  GeometryNameIni(.csm), GeometryNamePints(.txt),
  Cycle, Coefficient are inputs for matchCSM function.

  1. GeometryNameIni(.csm) is the geometry model that contains
    the initial guess of design parameters. Some examples of
```

```
   GeometryNameIni.csm can be found in Demo folder.
   Example: ESPMatchCSM/Demo/Box/boxIni.csm

2. GeometryNamePints(.txt) is the cloud of points file that
   will need be fitted. The format of it should be 3 columns
   in txt file. The first column are the X coordinates of
   points, the second coloum are the Y coordinates of the
   points, the third column are the Z coordinates of the
   points. This file can be provided by user, or generated
   in the 2.1 step. Some examples of GeometryNamePints.txt
   can be found in Demo folder.
   Example: ESPMatchCSM/Demo/Box/boxPints.csm

3. Cycle is the maximum number of cycles during running
   the fitting algorithm. The larger this number, the
   more accurte of the fitting reuslts, however, the larger
   this number, the more running time will be needed.
   Generally, for the sample geometry, this number
   could be set as 10, for the complate geometry,
   this number could be set as 15~20.

4. Cofficient is the number that will be used in
   classification technique. When it is set as larger
   than 1, durning the first several fitting cycles,
   the points in the junction between two components
   will not be count. This reduce the mis-classification
   problem. But for the geometry that contains small parts,
```

```
    large this number will lead to insufficient points on
    small faces. Generally, for the sample geometry,
    this number could be set as 2, for the complate geometry,
    this number could be set as 1~1.2.
2.2.3 Output results
  After running this process, fitting reuslts for the
  cloud of points file will be generated in the bin
  folder and named as
  1. "Ini_Parameters"     : initial guess of design parameters
  2. "InitialB"           : discrete of points from initial
                            design parameters
  3. "InitialA"           : discrete of points after
                            initialization
  4. "Result_Fitting"     : discrete of points from fitting result
                            design parameters
  5. "Result_Parameters1" : result design parameters after
                            1 cycle fitting
  6. "Result_ParametersF" : result design parameters after total
                            fitting process
  7. "RMS"                : RMS of distances history
  8. "Result"             : result of design parameters and
                            running time analysis
```

Appendix C

Appendix: Example of the Result File for MatchCSM

```
*****************************************************************
*                                                               *
*                   Finished the LM calculation                 *
*           This is the finial result of the optimization       *
*                   fuselage6 Fitting Problem                   *
*                                                               *
*****************************************************************

==> Total number of points in cloud is              6114
==> [Maximum Cycles, Cofficient Classification]   [   15,     2.0000000]
==> Total number of cycles for LM is                  15
==> Total number of iterations for LM is             146
```

```
==> Total running time for LM is             141.6911080
==> Per Iteration time for LM is               0.8609382
==> Per Iteration time for LM(NoBuild) is      0.5477885
==> ocsmBuild time in per Iteration is         0.3131497
==> The finial  RMS is                         0.0012419


*****************************************************************
*                                                               *
*                  Singal Running Time Analysis                 *
*       This includes singal running time for each function;    *
*          and also the total running time per Iteration.       *
*                                                               *
*****************************************************************


=====> Once time for each part <=====
==> per ocsmBuild(init) time is       0.0843590
==> per tess(init) time is            0.1282840
==> per initial/class time is         0.1199840
==> per ocsmBuild(Jaco) time is       0.0097040
==> per LM function time is           0.0608990
==> per ocsmBuild(attr) time is       0.0850650
==> per attribution time is           0.0049290
==> per objective time is             0.0000620
==> per ocsmGetVel(Jaco) time is      0.0162490
==> per EG_evaluate(Jaco) time is     0.0055440
==> per LM update rule time is        0.0000010
```

```
=====> Average once time for each part <=====
==> average per ocsmBuild(init) time is       0.0816946
==> average per tess(init) time is            0.1136664
==> average per initial/class time is         0.1172431
==> average per ocsmBuild(Jaco) time is       0.0096186
==> average per LM function time is           0.0899018
==> average per ocsmBuild(attr) time is       0.0823027
==> average per attribution time is           0.0061358
==> average per objective time is             0.0000988
==> average per ocsmGetVel(Jaco) time is      0.0185507
==> average per EG_evaluate(Jaco) time is     0.0060615
==> average per LM update rule time is        0.0003739
==> average per Iteration time is             0.8609382
==> average per Iteration(NoBuild) time is    0.5477885

**************************************************************
*                                                            *
*                  Total Running Time Analysis               *
*       This includes total running time for each function;  *
*       and also the total running time for each algorithm.  *
*                       Algorithm includes:                  *
*                  1. Initialization/Classification          *
*                  2. Jacobian                               *
*                  3. Levenberg Marquart                     *
*                  4. Attribution                            *
*                                                            *
**************************************************************
```

```
=====> Total time for each part <=====
==> total ocsmBuild(init) time is       1.2254190
==> total tess(init) time is            1.7049960
==> total initial/class time is         1.7586460
==> total ocsmBuild(Jaco) time is      33.7036740
==> total LM function time is          13.1256680
==> total ocsmBuild(attr) time is      12.0161890
==> total attribution time is           0.8958210
==> total objective time is             0.0144210
==> total ocsmGetVel(Jaco) time is     65.0016470
==> total EG_evaluate(Jaco) time is     0.8849760
==> total LM update rule time is        0.0545840

=====> Total time for each algorithm <=====
==> totoal Initi/class time verify  =     4.6890610
==> totoal Jacobian time verify     =    99.5902970
==> totoal LM time verify           =    13.1946730
==> totoal attribute time verify    =    12.9120100
==> totoal others time              =    11.3050670

=====> Total time for each algorithm (No ocsmBuild) <======
==> totoal Initi/class time verify  =     3.4636420
==> totoal Jacobian time verify     =    65.8866230
==> totoal LM time verify           =    13.1946730
==> totoal attribute time verify    =     0.8958210
==> totoal others time              =    11.3050670
```

```
==> totoal ocsmBuild time           =    46.9452820

==> totoal time without ocsmBuild   =    94.7458260


*****************************************************************

*                                                               *

*                   Result of Design Parameters                 *

*                                                               *

*****************************************************************

==> [    Initial,      Result]

==> [    0.1000000,    0.1017897]

==> [    2.1000000,    0.9987739]

==> [    2.6000000,    1.5999559]

==> [    2.6000000,    1.6000878]

==> [    1.8000000,    1.0016746]

==> [    0.1000000,    0.7988748]

==> [    0.1000000,    0.0969182]

==> [    2.1000000,    0.9974154]

==> [    2.6000000,    2.0001656]

==> [    2.6000000,    2.0000629]

==> [    1.8000000,    1.2031732]

==> [    0.1000000,    0.3985131]

==> [    0.0000000,    0.0000000]

==> [    2.0000000,    1.0000000]

==> [    5.0000000,    4.0000000]

==> [    9.0000000,    8.0000000]

==> [   14.0000000,   12.0000000]

==> [   18.0000000,   15.9986388]
```

```
==> [    0.0000000,    0.0005694]

==> [    0.2000000,    0.0998148]

==> [    0.5000000,    0.4000871]

==> [    0.6000000,    0.3999738]

==> [    0.4000000,    0.3003751]

==> [    0.3000000,    0.1997758]
```

Bibliography

[1] Nirant V. Puntambekar, Andrei G. Jablokow, and H.Joseph Sommer III. Unified review of 3d model generation for reverse engineering. *Computer Integrated Manufacturing Systems*, 7(4):259 – 268, 1994. ISSN 0951-5240. doi: http://dx.doi.org/10.1016/0951-5240(94)90015-9. URL http://www.sciencedirect.com/science/article/pii/0951524094900159.

[2] ZHOU Min. A new approach of composite surface reconstruction based on reverse engineering. *Procedia Engineering*, 23(0):594 – 599, 2011. ISSN 1877-7058. doi: http://dx.doi.org/10.1016/j.proeng.2011.11.2552. URL http://www.sciencedirect.com/science/article/pii/S1877705811053951. {PEEA} 2011.

[3] Roseline Beniere, Gerard Subsol, Gilles Gesquiere, Francois Le Breton, and William Puech. A comprehensive process of reverse engineering from 3d meshes to {CAD} models. *Computer-Aided Design*, 45(11):1382 – 1393, 2013. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2013.06.004. URL http://www.sciencedirect.com/science/article/pii/S0010448513001012.

[4] Beniere R, Subsol G, Gesquiere G, Le Breton F, and Puech W. Recovering primitives in 3d cad meshes. *SPIE electronic imaging 2011, 3D imaging, interaction and measurement*, 7864:1–9, 2011.

[5] Wenlei Xiao, Lianyu Zheng, Ji Huan, and Pei Lei. A complete cad/cam/cnc solution for step-compliant manufacturing. *Robotics and Computer-Integrated Manufacturing*, 31(0):1 – 10, 2015. ISSN 0736-5845. doi: http://dx.doi.org/10.1016/j.rcim.2014.06.003. URL http://www.sciencedirect.com/science/article/pii/S073658451400043X.

[6] T Wu, F Portheine, A Popovic, P Bast, M Wehmoeller, and K Radermacher. An interface for the data exchange between {CAS} and cad/cam systems. *International Congress Series*, 1256(0):703 – 709, 2003. ISSN 0531-5131. doi: http://dx.doi.org/10.1016/S0531-5131(03)00282-6. URL http://www.sciencedirect.com/science/article/pii/S0531513103002826. {CARS} 2003. Computer Assisted Radiology and Surgery. Proceedings of the 17th International Congress and Exhibition.

[7] Michael J. Pratt, Bill D. Anderson, and Tony Ranger. Towards the standardized exchange of parameterized feature-based {CAD} models. *Computer-Aided Design*, 37(12):1251 – 1265, 2005. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2004.12.005. URL http://www.sciencedirect.com/science/article/pii/S0010448505000035.

[8] Hiroki Yoshihara, Tatsuya Yoshii, Tadahiro Shibutani, and Takashi Maekawa. Topologically robust b-spline surface reconstruction from point clouds using level set methods and iterative geometric fitting algorithms. *Computer Aided Geometric Design*, 29(7):422 – 434, 2012. ISSN 0167-8396. doi: http://dx.doi.org/10.1016/j.cagd.2012.03.007. URL http://www.sciencedirect.com/science/article/pii/S0167839612000337. Geometric Modeling and Processing 2012.

[9] Ulrich Bauer and Konrad Polthier. Generating parametric models of tubes from laser scans. *Computer-Aided Design*, 41(10):719 – 729, 2009. ISSN 0010-4485. doi:

http://dx.doi.org/10.1016/j.cad.2009.01.002. URL http://www.sciencedirect.com/science/article/pii/S0010448509000177. Selected Papers from the 2007 New Advances in Shape Analysis and Geometric Modeling Workshop.

[10] Lip M. Lai and Matthew M.F. Yuen. Blending of mesh objects to parametric surface. *Computers and Graphics*, 46(0):283 – 293, 2015. ISSN 0097-8493. doi: http://dx.doi.org/10.1016/j.cag.2014.09.030. URL http://www.sciencedirect.com/science/article/pii/S0097849314001186.

[11] Akemi Galvez and Andres Iglesias. A new iterative mutually coupled hybrid ga-pso approach for curve fitting in manufacturing. *Applied Soft Computing*, 13(3):1491 – 1504, 2013. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2012.05.030. URL http://www.sciencedirect.com/science/article/pii/S1568494612002918. Hybrid evolutionary systems for manufacturing processes.

[12] Hongwei Lin. Adaptive data fitting by the progressive-iterative approximation. *Computer Aided Geometric Design*, 29(7):463 – 473, 2012. ISSN 0167-8396. doi: http://dx.doi.org/10.1016/j.cagd.2012.03.005. URL http://www.sciencedirect.com/science/article/pii/S0167839612000313. Geometric Modeling and Processing 2012.

[13] Wenni Zheng, Pengbo Bo, Yang Liu, and Wenping Wang. Fast b-spline curve fitting by l-bfgs. *Computer Aided Geometric Design*, 29(7):448 – 462, 2012. ISSN 0167-8396. doi: http://dx.doi.org/10.1016/j.cagd.2012.03.004. URL http://www.sciencedirect.com/science/article/pii/S0167839612000301. Geometric Modeling and Processing 2012.

[14] Zhang Lei, Gu Tianqi, Zhao Ji, Ji Shijun, Sun Qingzhou, and Hu Ming. An adaptive moving total least squares method for curve fitting. *Measurement*, 49(0):107 – 112, 2014. ISSN 0263-2241. doi: http://dx.doi.org/10.1016/

j.measurement.2013.11.050. URL http://www.sciencedirect.com/science/article/pii/S0263224113006039.

[15] Xiangchao Zhang, Hao Zhang, Xiaoying He, Min Xu, and Xiangqian Jiang. Chebyshev fitting of complex surfaces for precision metrology. *Measurement*, 46(9):3720 – 3724, 2013. ISSN 0263-2241. doi: http://dx.doi.org/10.1016/j.measurement.2013.04.017. URL http://www.sciencedirect.com/science/article/pii/S0263224113001243.

[16] Simon Flory. Fitting curves and surfaces to point clouds in the presence of obstacles. *Computer Aided Geometric Design*, 26(2):192 – 202, 2009. ISSN 0167-8396. doi: http://dx.doi.org/10.1016/j.cagd.2008.04.003. URL http://www.sciencedirect.com/science/article/pii/S0167839608000289.

[17] Jui-Sheng Chou, Min-Yuan Cheng, Yu-Wei Wu, and Anh-Duc Pham. Optimizing parameters of support vector machine using fast messy genetic algorithm for dispute classification. *Expert Systems with Applications*, 41(8):3955 – 3964, 2014. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2013.12.035. URL http://www.sciencedirect.com/science/article/pii/S0957417413010105.

[18] Damir Vucina, Zeljan Lozina, and Igor Pehnec. Computational procedure for optimum shape design based on chained bezier surfaces parameterization. *Engineering Applications of Artificial Intelligence*, 25(3):648 – 667, 2012. ISSN 0952-1976. doi: http://dx.doi.org/10.1016/j.engappai.2011.11.007. URL http://www.sciencedirect.com/science/article/pii/S0952197611002181.

[19] Michael Athanasopoulos, Hassan Ugail, and Gabriela Gonzalez Castro. Parametric design of aircraft geometry using partial differential equations. *Advances in Engineering Software*, 40(7):479 – 486, 2009. ISSN 0965-9978. doi: http://dx.doi.org/10.

1016/j.advengsoft.2008.08.001. URL http://www.sciencedirect.com/science/article/pii/S0965997808001531.

[20] Jonathan Byrne, Philip Cardiff, Anthony Brabazon, and Michael ONeill. Evolving parametric aircraft models for design exploration and optimisation. *Neurocomputing*, 142(0):39 – 47, 2014. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2014.04.004. URL http://www.sciencedirect.com/science/article/pii/S092523121400530X. {SI} Computational Intelligence Techniques for New Product Development.

[21] Christof Ledermann, Claus Hanske, Jorg Wenzel, Paolo Ermanni, and Roland Kelm. Associative parametric {CAE} methods in the aircraft pre-design. *Aerospace Science and Technology*, 9(7):641 – 651, 2005. ISSN 1270-9638. doi: http://dx.doi.org/10.1016/j.ast.2005.05.001. URL http://www.sciencedirect.com/science/article/pii/S1270963805000726.

[22] Jinggao Li, Byung Chul Kim, and Soonhung Han. Parametric exchange of round shapes between a mechanical {CAD} system and a ship {CAD} system. *Computer-Aided Design*, 44(2):154 – 161, 2012. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2011.08.003. URL http://www.sciencedirect.com/science/article/pii/S0010448511001916.

[23] Pengbo Bo, Ruotian Ling, and Wenping Wang. A revisit to fitting parametric surfaces to point clouds. *Computers and Graphics*, 36(5):534 – 540, 2012. ISSN 0097-8493. doi: http://dx.doi.org/10.1016/j.cag.2012.03.036. URL http://www.sciencedirect.com/science/article/pii/S0097849312000751. Shape Modeling International (SMI) Conference 2012.

[24] Peter J. Bickel and Kjell A. Doksum. Mathematical statistics: Basic and selected topics. *Pearson Prentice-Hall*, 1, 2001.

[25] Nau D. Mantyla M. and Shah J. Challenges in feature based manufacturing research. *Communications of the ACM*, 39:77–85, 1996.

[26] Tamas Varady, Ralph R Martin, and Jordan Cox. Reverse engineering of geometric models–an introduction. *Computer-Aided Design*, 29(4):255 – 268, 1997. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/S0010-4485(96)00054-1. URL http://www.sciencedirect.com/science/article/pii/S0010448596000541. Reverse Engineering of Geometric Models.

[27] Ping-Yi Chao and Yu chou Wang. A data exchange framework for networked cad/cam. *Computers in Industry*, 44(2):131 – 140, 2001. ISSN 0166-3615. doi: http://dx.doi.org/10.1016/S0166-3615(00)00082-8. URL http://www.sciencedirect.com/science/article/pii/S0166361500000828.

[28] H. Pottmann, S. Leopoldseder, M. Hofer, T. Steiner, and W. Wang. Industrial geometry: recent advances and applications in {CAD}. *Computer-Aided Design*, 37(7):751 – 766, 2005. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2004.08.013. URL http://www.sciencedirect.com/science/article/pii/S0010448504001988.

[29] Hai-Jun Rong, Ya-Xin Jia, and Guang-She Zhao. Aircraft recognition using modular extreme learning machine. *Neurocomputing*, 128(0):166 – 174, 2014. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2012.12.064. URL http://www.sciencedirect.com/science/article/pii/S0925231213010023.

[30] Saeid Motavalli. Review of reverse engineering approaches. *Computers and Industrial Engineering*, 35(1-2):25 – 28, 1998. ISSN 0360-8352. doi: http://dx.doi.org/10.1016/S0360-8352(98)00011-4. URL http://www.sciencedirect.com/science/article/pii/S0360835298000114.

[31] N. Werghi, R. Fisher, C. Robertson, and A. Ashbrook. Object reconstruction by incorporating geometric constraints in reverse engineering. *Computer-Aided Design*, 31(6):363 – 399, 1999. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/S0010-4485(99)00038-X. URL http://www.sciencedirect.com/science/article/pii/S001044859900038X.

[32] Jianbing Huang and Chia-Hsiang Menq. Automatic cad model reconstruction from multiple point clouds for reverse engineering. *Journal of Computing and Information Science in Engineering*, 2(3):160 – 170, 2003. ISSN 1530-9827. doi: 10.1115/1.1529210. URL http://dx.doi.org/10.1115/1.1529210.

[33] Syed Afaq Ali Shah, Mohammed Bennamoun, and Farid Boussaid. A novel feature representation for automatic 3d object recognition in cluttered scenes. *Neurocomputing*, 205:1 – 15, 2016. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2015.11.019. URL http://www.sciencedirect.com/science/article/pii/S0925231215017385.

[34] Duc Fehr, William J. Beksi, Dimitris Zermas, and Nikolaos Papanikolopoulos. Covariance based point cloud descriptors for object detection and recognition. *Computer Vision and Image Understanding*, 142:80 – 93, 2016. ISSN 1077-3142. doi: http://dx.doi.org/10.1016/j.cviu.2015.06.008. URL http://www.sciencedirect.com/science/article/pii/S1077314215001368.

[35] Weisheng Li, Peng Dong, Bin Xiao, and Lifang Zhou. Object recognition based on the region of interest and optimal bag of words model. *Neurocomputing*, 172:271 – 280, 2016. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2015.01.083. URL http://www.sciencedirect.com/science/article/pii/S0925231215005925.

[36] Junhwan Kim, Michael J. Pratt, Raj G. Iyer, and Ram D. Sriram. Standardized data exchange of {CAD} models with design intent. *Computer-Aided Design*, 40(7):760 – 777, 2008. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2007.06.014. URL http://www.sciencedirect.com/science/article/pii/S0010448507001625. Current State and Future of Product Data Technologies (PDT).

[37] H. Tsige-Tamirat, U. Fischer, A. Serikov, and S. Stickel. Use of mccad for the conversion of {ITER} {CAD} data to {MCNP} geometry. *Fusion Engineering and Design*, 83(10-12):1771 – 1773, 2008. ISSN 0920-3796. doi: http://dx.doi.org/10.1016/j.fusengdes.2008.07.040. URL http://www.sciencedirect.com/science/article/pii/S092037960800238X. Proceedings of the Eight International Symposium of Fusion Nuclear Technology ISFNT-8 {SI}.

[38] Byung Chul Kim, Duhwan Mun, Soonhung Han, and Michael J. Pratt. A method to exchange procedurally represented 2d {CAD} model data using {ISO} 10303 {STEP}. *Computer-Aided Design*, 43(12):1717 – 1728, 2011. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2011.07.006. URL http://www.sciencedirect.com/science/article/pii/S0010448511001837.

[39] Sang-Uk Cheon, Byung Chul Kim, Duhwan Mun, and Soonhung Han. A procedural method to exchange editable 3d data from a free-hand 2d sketch modeling system into 3d mechanical {CAD} systems. *Computer-Aided Design*, 44(2):123 – 131, 2012. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2011.10.003. URL http://www.sciencedirect.com/science/article/pii/S0010448511002624.

[40] J Whyte, N Bouchlaghem, A Thorpe, and R McCaffer. From {CAD} to virtual reality: modelling approaches, data exchange and interactive 3d building design tools. *Automation in Construction*, 10(1):43 – 55, 2000. ISSN 0926-5805. doi: http:

//dx.doi.org/10.1016/S0926-5805(99)00012-6. URL http://www.sciencedirect.com/science/article/pii/S0926580599000126.

[41] V. Stamati, G. Antonopoulos, Ph. Azariadis, and I. Fudos. A parametric feature-based approach to reconstructing traditional filigree jewelry. *Computer-Aided Design*, 43(12):1814 – 1828, 2011. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2011.07.002. URL http://www.sciencedirect.com/science/article/pii/S0010448511001692.

[42] Sebastian Ochmann, Richard Vock, Raoul Wessel, and Reinhard Klein. Automatic reconstruction of parametric building models from indoor point clouds. *Computers & Graphics*, 54:94 – 103, 2016. ISSN 0097-8493. doi: http://dx.doi.org/10.1016/j.cag.2015.07.008. URL http://www.sciencedirect.com/science/article/pii/S0097849315001119. Special Issue on CAD/Graphics 2015.

[43] Jiju Peethambaran and Ramanathan Muthuganapathy. Reconstruction of water-tight surfaces through delaunay sculpting. *Computer-Aided Design*, 58 (0):62 – 72, 2015. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2014.08.021. URL http://www.sciencedirect.com/science/article/pii/S0010448514001900. Solid and Physical Modeling 2014.

[44] M. Sevaux and Y. Mineur. A curve-fitting genetic algorithm for a styling application. *European Journal of Operational Research*, 179(3):895 – 905, 2007. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2005.03.065. URL http://www.sciencedirect.com/science/article/pii/S0377221705007642.

[45] Xiuyang Zhao, Caiming Zhang, Li Xu, Bo Yang, and Zhiquan Feng. Iga-based point cloud fitting using b-spline surfaces for reverse engineering. *Information Sciences*, 245(0):276 – 289, 2013. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/

j.ins.2013.04.022. URL http://www.sciencedirect.com/science/article/pii/S0020025513003198. Statistics with Imperfect Data.

[46] Chongyang Deng and Hongwei Lin. Progressive and iterative approximation for least squares b-spline curve and surface fitting. *Computer-Aided Design*, 47(0):32 – 44, 2014. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2013.08.012. URL http://www.sciencedirect.com/science/article/pii/S0010448513001528.

[47] Yuki Kineri, Mingsi Wang, Hongwei Lin, and Takashi Maekawa. B-spline surface fitting by iterative geometric interpolation/approximation algorithms. *Computer-Aided Design*, 44(7):697 – 708, 2012. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2012.02.011. URL http://www.sciencedirect.com/science/article/pii/S0010448512000528.

[48] James Andrews and Carlo H. Sequin. Generalized, basis-independent kinematic surface fitting. *Computer-Aided Design*, 45(3):615 – 620, 2013. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2012.10.047. URL http://www.sciencedirect.com/science/article/pii/S0010448512002667.

[49] Edgar Janunts, Marc KannengieBer, and Achim Langenbucher. Parametric fitting of corneal height data to a biconic surface. *Zeitschrift fur Medizinische Physik*, 1(0):–, 2014. ISSN 0939-3889. doi: http://dx.doi.org/10.1016/j.zemedi.2014.02.005. URL http://www.sciencedirect.com/science/article/pii/S0939388914000324.

[50] N. Venkaiah and M.S. Shunmugam. Evaluation of form data using computational geometric techniques-part ii: Cylindricity error. *International Journal of Machine Tools and Manufacture*, 47(7–8):1237 – 1245, 2007. ISSN 0890-6955. doi: http://dx.doi.org/10.1016/j.ijmachtools.2006.08.011. URL http://www.sciencedirect.com/science/article/pii/S0890695506002069.

[51] A.M. Malyscheff, T.B. Trafalis, and S. Raman. From support vector machine learning to the determination of the minimum enclosing zone. *Computers and Industrial Engineering*, 42(1):59 – 74, 2002. ISSN 0360-8352. doi: http://dx.doi.org/10.1016/S0360-8352(02)00003-7. URL http://www.sciencedirect.com/science/article/pii/S0360835202000037.

[52] Tohru Kanada. Evaluation of spherical form errors-computation of sphericity by means of minimum zone method and some examinations with using simulated data. *Precision Engineering*, 17(4):281 – 289, 1995. ISSN 0141-6359. doi: http://dx.doi.org/10.1016/0141-6359(95)00017-8. URL http://www.sciencedirect.com/science/article/pii/0141635995000178.

[53] Kenichi Kanatani and Yasuyuki Sugaya. Performance evaluation of iterative geometric fitting algorithms. *Computational Statistics and Data Analysis*, 52 (2):1208 – 1222, 2007. ISSN 0167-9473. doi: http://dx.doi.org/10.1016/j.csda.2007.05.013. URL http://www.sciencedirect.com/science/article/pii/S0167947307002150.

[54] Simon Flory and Michael Hofer. Constrained curve fitting on manifolds. *Computer-Aided Design*, 40(1):25 – 34, 2008. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2007.01.012. URL http://www.sciencedirect.com/science/article/pii/S0010448507000280. Constrained Design of Curves and Surfaces.

[55] Jun Wang and Zeyun Yu. Quadratic curve and surface fitting via squared distance minimization. *Computers and Graphics*, 35(6):1035 – 1050, 2011. ISSN 0097-8493. doi: http://dx.doi.org/10.1016/j.cag.2011.09.001. URL http://www.sciencedirect.com/science/article/pii/S0097849311001464.

[56] Simon Flory and Michael Hofer. Surface fitting and registration of point clouds using approximations of the unsigned distance function. *Computer Aided Geometric*

Design, 27(1):60 – 77, 2010. ISSN 0167-8396. doi: http://dx.doi.org/10.1016/j.cagd.2009.09.001. URL http://www.sciencedirect.com/science/article/pii/S0167839609000971.

[57] L. Pourkarimi, M.A. Yaghoobi, and M. Mashinchi. Efficient curve fitting: An application of multiobjective programming. *Applied Mathematical Modelling*, 35(1):346 – 365, 2011. ISSN 0307-904X. doi: http://dx.doi.org/10.1016/j.apm.2010.06.009. URL http://www.sciencedirect.com/science/article/pii/S0307904X10002374.

[58] Charlie C.L. Wang and Gershon Elber. Multi-dimensional dynamic programming in ruled surface fitting. *Computer-Aided Design*, 51(0):39 – 49, 2014. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2014.02.004. URL http://www.sciencedirect.com/science/article/pii/S0010448514000281.

[59] D. Chaudhuri and A. Samal. A simple method for fitting of bounding rectangle to closed regions. *Pattern Recognition*, 40(7):1981 – 1989, 2007. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/j.patcog.2006.08.003. URL http://www.sciencedirect.com/science/article/pii/S0031320306003530.

[60] Quan Wang and Kim L. Boyer. The active geometric shape model: A new robust deformable shape model and its applications. *Computer Vision and Image Understanding*, 116(12):1178 – 1194, 2012. ISSN 1077-3142. doi: http://dx.doi.org/10.1016/j.cviu.2012.08.004. URL http://www.sciencedirect.com/science/article/pii/S1077314212001154.

[61] Akemi Galvez and Andres Iglesias. Efficient particle swarm optimization approach for data fitting with free knot -splines. *Computer-Aided Design*, 43(12):1683 – 1692, 2011. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2011.07.010. URL http://www.sciencedirect.com/science/article/pii/S0010448511001874.

[62] Akemi Galvez and Andres Iglesias. Particle swarm optimization for non-uniform rational b-spline surface reconstruction from clouds of 3d data points. *Information Sciences*, 192(0):174 – 192, 2012. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2010.11.007. URL http://www.sciencedirect.com/science/article/pii/S0020025510005529. Swarm Intelligence and Its Applications.

[63] Hongmei Kang, Falai Chen, Yusheng Li, Jiansong Deng, and Zhouwang Yang. Knot calculation for spline fitting via sparse optimization. *Computer-Aided Design*, 58(0):179 – 188, 2015. ISSN 0010-4485. doi: http://dx.doi.org/10.1016/j.cad.2014.08.022. URL http://www.sciencedirect.com/science/article/pii/S0010448514001912. Solid and Physical Modeling 2014.

[64] Emilio Carrizosa and Dolores Romero Morales. Supervised classification and mathematical optimization. *Computers and Operations Research*, 40(1):150 – 165, 2013. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2012.05.015. URL http://www.sciencedirect.com/science/article/pii/S0305054812001190.

[65] Yang Liu and Masatoshi Shimoda. Non-parametric shape optimization method for natural vibration design of stiffened shells. *Computers and Structures*, 146 (0):20 – 31, 2015. ISSN 0045-7949. doi: http://dx.doi.org/10.1016/j.compstruc.2014.08.003. URL http://www.sciencedirect.com/science/article/pii/S0045794914001850.

[66] Sahibsingh A. Dudani, Kenneth J. Breeding, and R.B. McGhee. Aircraft identification by moment invariants. *Computers, IEEE Transactions on*, C-26(1):39–46, Jan 1977. ISSN 0018-9340. doi: 10.1109/TC.1977.5009272.

[67] Feng Zhang, Shang qian Liu, Da bao Wang, and Wei Guan. Aircraft recognition in infrared image using wavelet moment invariants. *Image and Vision Computing*, 27(4):313 – 318, 2009. ISSN 0262-8856. doi: http://dx.doi.org/10.1016/

j.imavis.2008.08.007. URL http://www.sciencedirect.com/science/article/pii/S0262885608001807.

[68] Jan Flusser, Tomas Suk, and Barbara Zitova. *Moments and Moment Invariants in Pattern Recognition*. John Wiley & Sons, Ltd, 2009. ISBN 9780470684757. doi: 10.1002/9780470684757.ch2. URL http://dx.doi.org/10.1002/9780470684757.ch2.

[69] Yiliang Zeng, Jinhui Lan, Chuanzhao Han, Kewei Huang, Jiehui Li, and Xuefei Shi. Aircraft recognition based on improved iterative threshold selection and skeleton zernike moment. *Optik - International Journal for Light and Electron Optics*, 125(14):3733 – 3737, 2014. ISSN 0030-4026. doi: http://dx.doi.org/10.1016/j.ijleo.2014.01.135. URL http://www.sciencedirect.com/science/article/pii/S003040261400309X.

[70] U.H. Augsdorfer, N.A. Dodgson, and M.A. Sabin. Artifact analysis on b-splines, box-splines and other surfaces defined by quadrilateral polyhedra. *Computer Aided Geometric Design*, 28(3):177 – 197, 2011. ISSN 0167-8396. doi: http://dx.doi.org/10.1016/j.cagd.2010.04.002. URL http://www.sciencedirect.com/science/article/pii/S0167839610000397.

[71] INFORMS Computing Society. The nature of mathematical programming. *Mathematical Programming Glossary*, 2010. URL http://glossary.computing.society.informs.org/ver2/mpgwiki/index.php?title=Extra:Mathematical_programming.

[72] Matthias Ehrgott. Multicriteria optimization. *Birkhauser*, 2012.

[73] David A. Van Veldhuisen Carlos A. Coello Coello, Gary B. Lamont. Evolutionary algorithms for solving multi-objective problems. *Springer*, 2012.

[74] Ching-Lai Hwang and Abu Syed Md Masud. *Multiple objective decision making, methods and applications: a state-of-the-art survey*. Springer-Verlag, 2012. ISBN 978-0-387-09111-2.

[75] Gerhard Venter. Review of optimization techniques. *Research Gate*, 2010.

[76] Jussi Hakanen. Introduction to unconstrained optimization - gradient-based methods, 2014. URL http://users.jyu.fi/~jhaka/opt/TIES483_unconstrained_gradient2.pdf.

[77] Banzhaf Wolfgang, Nordin Peter, Keller Robert, and Francone Frank. *Genetic Programming ? An Introduction*. Morgan Kaufmann, 1998. ISBN 978-1558605107.

[78] Lin Wang, Bo Yang, and Jeff Orchard. Particle swarm optimization using dynamic tournament topology. *Applied Soft Computing*, 48:584 – 596, 2016. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2016.07.041. URL http://www.sciencedirect.com/science/article/pii/S1568494616303763.

[79] Yudong Zhang, Shuihua Wang, , and Genlin Ji. A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, 2015:38, 2015. doi: http://dx.doi.org/10.1155/2015/931256. URL http://dx.doi.org/10.1155/2015/931256.

[80] John McCullock. Introduction to the particle swarm optimization, 2012. URL http://mnemstudio.org/particle-swarm-introduction.htm.

[81] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[82] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21: 1087–1092, 1953.

[83] Yinglong Wang, Guangle Bu, Yongkun Wang, Tingran Zhao, Zhen Zhang, and Zhaoyou Zhu. Application of a simulated annealing algorithm to design and optimize a pressure-swing distillation process. *Computers & Chemical Engineering*, 95:97 – 107, 2016. ISSN 0098-1354. doi: http://dx.doi.org/10.1016/j.compchemeng.2016.09.014. URL http://www.sciencedirect.com/science/article/pii/S0098135416303003.

[84] Granville V., Krivanek M., and Rasson. Simulated annealing: A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6): 652?656, 1994. doi: doi:10.1109/34.295910.

[85] Xuewu Du, Peng Zhang, and Wenya Ma. Some modified conjugate gradient methods for unconstrained optimization. *Journal of Computational and Applied Mathematics*, 305:92 – 114, 2016. ISSN 0377-0427. doi: http://dx.doi.org/10.1016/j.cam.2016.04.004. URL http://www.sciencedirect.com/science/article/pii/S0377042716301686.

[86] Yaohua Hu, Xiaoqi Yang, and Chee-Khian Sim. Inexact subgradient methods for quasi-convex optimization problems. *European Journal of Operational Research*, 240(2):315 – 327, 2015. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2014.05.017. URL http://www.sciencedirect.com/science/article/pii/S037722171400424X.

[87] Armin Ataei and Qian Wang. A probabilistic ellipsoid algorithm for linear optimization problems with uncertain {LMI} constraints. *Automatica*, 52:248 – 254, 2015. ISSN 0005-1098. doi: http://dx.doi.org/10.1016/j.automatica.2014.11.010. URL http://www.sciencedirect.com/science/article/pii/S000510981400569X.

[88] J. Sobieszczanski-Sobieski and R.T. Haftka. Multidisciplinary aerospace design optimization: survey of recent developments. *Struct. Multidiscip. Optim.*, 14(1): 1–23, 1997.

[89] Shuangcheng Sun, Hong Qi, Fangzhou Zhao, Liming Ruan, and Bingxi Li. Inverse geometry design of two-dimensional complex radiative enclosures using krill herd optimization algorithm. *Applied Thermal Engineering*, 98:1104 – 1115, 2016. ISSN 1359-4311. doi: http://dx.doi.org/10.1016/j.applthermaleng.2016.01.017. URL http://www.sciencedirect.com/science/article/pii/S1359431116000727.

[90] Zhenyu Yang, B. Sendhoff, Ke Tang, and Xin Yao. Target shape design optimization by evolving b-splines with cooperative coevolution. *Applied Soft Computing*, 48:672 – 682, 2016. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2016.07.027. URL http://www.sciencedirect.com/science/article/pii/S1568494616303532.

[91] S. Shaaban. Aero-economical optimization of wells turbine rotor geometry. *Energy Conversion and Management*, 126:20 – 31, 2016. ISSN 0196-8904. doi: http://dx.doi.org/10.1016/j.enconman.2016.07.068. URL http://www.sciencedirect.com/science/article/pii/S019689041630646X.

[92] Paola Boito and Roberto Grena. Optimization of the geometry of fresnel linear collectors. *Solar Energy*, 135:479 – 486, 2016. ISSN 0038-092X. doi: http://dx.doi.org/10.1016/j.solener.2016.05.060. URL http://www.sciencedirect.com/science/article/pii/S0038092X16301785.

[93] Xiaobin Cui, Jingxia Guo, and Jianxin Zheng. Optimization of geometry parameters for ceramic cutting tools in intermittent turning of hardened steel. *Materials & Design*, 92:424 – 437, 2016. ISSN 0264-1275. doi: http://dx.doi.org/10.1016/

j.matdes.2015.12.089. URL http://www.sciencedirect.com/science/article/pii/S0264127515309473.

[94] Saeed Eini, Hamid Reza Shahhosseini, Majid Javidi, Mahdi Sharifzadeh, and Davood Rashtchian. Inherently safe and economically optimal design using multi-objective optimization: The case of a refrigeration cycle. *Process Safety and Environmental Protection*, 104, Part A:254 – 267, 2016. ISSN 0957-5820. doi: http://dx.doi.org/10.1016/j.psep.2016.09.010. URL http://www.sciencedirect.com/science/article/pii/S0957582016302099.

[95] Nader Azad, Elkafi Hassini, and Manish Verma. Disruption risk management in railroad networks: An optimization-based methodology and a case study. *Transportation Research Part B: Methodological*, 85:70 – 88, 2016. ISSN 0191-2615. doi: http://dx.doi.org/10.1016/j.trb.2016.01.001. URL http://www.sciencedirect.com/science/article/pii/S0191261516000059.

[96] Matt Thompson. Natural gas storage valuation, optimization, market and credit risk management. *Journal of Commodity Markets*, pages –, 2016. ISSN 2405-8513. doi: http://dx.doi.org/10.1016/j.jcomm.2016.07.004. URL http://www.sciencedirect.com/science/article/pii/S2405851315300167.

[97] Giovanni Margarido Righetto, Reinaldo Morabito, and Douglas Alem. A robust optimization approach for cash flow management in stationery companies. *Computers & Industrial Engineering*, 99:137 – 152, 2016. ISSN 0360-8352. doi: http://dx.doi.org/10.1016/j.cie.2016.07.010. URL http://www.sciencedirect.com/science/article/pii/S0360835216302352.

[98] Carlos Gamarra and Josep M. Guerrero. Computational optimization techniques applied to microgrids planning: A review. *Renewable and Sustainable Energy Reviews*, 48:413 – 424, 2015. ISSN 1364-0321. doi: http://dx.doi.org/10.1016/j.

rser.2015.04.025. URL http://www.sciencedirect.com/science/article/pii/S1364032115002956.

[99] Masoud Zebarjadi and Alireza Askarzadeh. Optimization of a reliable grid-connected pv-based power plant with/without energy storage system by a heuristic approach. *Solar Energy*, 125:12 – 21, 2016. ISSN 0038-092X. doi: http://dx.doi.org/10.1016/j.solener.2015.11.045. URL http://www.sciencedirect.com/science/article/pii/S0038092X15006684.

[100] G. Mendes, C. Ioakimidis, and P. Ferrao. On the planning and analysis of integrated community energy systems: a review and survey of available tools. *Renew Sustain Energy Rev.*, 15:4836?4854, 2011.

[101] Vafaei M and Kazerani M. Optimal unit-sizing of a wind-hydrogen-diesel microgrid system for a remote community. *2011 IEEE Trondheim PowerTech*, page 1?7, 2011.

[102] N. Augustine and S. Suresh. Economic dispatch for a microgrid considering renewable energy cost functions. *Innov Smart Grid*, page 1?7, 2012.

[103] D. E. Olivares, C. A. Canizares, and M. Kazerani. A centralized energy management system for isolated microgrids. *IEEE Transactions on Smart Grid*, 5(4):1864–1875, July 2014. ISSN 1949-3053. doi: 10.1109/TSG.2013.2294187.

[104] Hong K. Lo and Anthony Chen. Traffic equilibrium problem with route-specific costs: formulation and algorithms. *Transportation Research*, Part B:493–513, 2000.

[105] Fausto Saleri Alfio Quarteroni and Paola Gervasio. *Scientific computing with MATLAB and Octave*. Springer, 2010. ISBN 978-3-642-45366-3.

[106] Griewank A and Walther A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008. ISBN 978-0898716597.

[107] Bechir Dali, Chong Li, and Jinhua Wang. Local convergence of newton's method on the heisenberg group. *Journal of Computational and Applied Mathematics*, 300:217 – 232, 2016. ISSN 0377-0427. doi: http://dx.doi.org/10.1016/j.cam.2015.12.025. URL http://www.sciencedirect.com/science/article/pii/S0377042715006378.

[108] Samir Adly, Huynh Van Ngai, and Van Vu Nguyen. Newton's method for solving generalized equations: Kantorovich's and smale's approaches. *Journal of Mathematical Analysis and Applications*, 439(1):396 – 418, 2016. ISSN 0022-247X. doi: http://dx.doi.org/10.1016/j.jmaa.2016.02.047. URL http://www.sciencedirect.com/science/article/pii/S0022247X16001815.

[109] Anton Evgrafov. State space newton's method for topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 278:272 – 290, 2014. ISSN 0045-7825. doi: http://dx.doi.org/10.1016/j.cma.2014.06.005. URL http://www.sciencedirect.com/science/article/pii/S004578251400190X.

[110] Ailei Lang, Zhanjie Song, Gaiyun He, and Yicun Sang. Profile error evaluation of free-form surface using sequential quadratic programming algorithm. *Precision Engineering*, 47:344 – 352, 2017. ISSN 0141-6359. doi: http://dx.doi.org/10.1016/j.precisioneng.2016.09.008. URL http://www.sciencedirect.com/science/article/pii/S0141635916302264.

[111] Emil Klintberg and Sebastien Gros. An inexact interior point method for optimizatioon of differential algebraic systems. *Computers & Chemical Engineering*, 92:163 – 171, 2016. ISSN 0098-1354. doi: http://dx.doi.org/10.1016/j.compchemeng.2016.04.013. URL http://www.sciencedirect.com/science/article/pii/S0098135416301090.

[112] Ji-Feng Bao, Chong Li, Wei-Ping Shen, Jen-Chih Yao, and Sy-Ming Guu. Approximate gauss-newton methods for solving underdetermined nonlinear least squares problems. *Applied Numerical Mathematics*, 111:92 – 110, 2017. ISSN 0168-9274. doi: http://dx.doi.org/10.1016/j.apnum.2016.08.007. URL http://www.sciencedirect.com/science/article/pii/S016892741630157X.

[113] D.W. Marquardt. An algorithm for least squares estimation of nonlinear parameters. *SIAM Journal*, 11:431–441, 1963.

[114] Jorge Nocedal and Stephen J. Wright. Numerical optimization, 1999.

[115] Mohammad Modarres and Ehsan Izadpanahi. Aggregate production planning by focusing on energy saving: A robust optimization approach. *Journal of Cleaner Production*, 133:1074 – 1085, 2016. ISSN 0959-6526. doi: http://dx.doi.org/10.1016/j.jclepro.2016.05.133. URL http://www.sciencedirect.com/science/article/pii/S0959652616306059.

[116] Xing Shi, Zhichao Tian, Wenqiang Chen, Binghui Si, and Xing Jin. A review on building energy efficient design optimization rom the perspective of architects. *Renewable and Sustainable Energy Reviews*, 65:872 – 884, 2016. ISSN 1364-0321. doi: http://dx.doi.org/10.1016/j.rser.2016.07.050. URL http://www.sciencedirect.com/science/article/pii/S136403211630377X.

[117] Shengwei Wang and Zhenjun Ma. Supervisory and optimal control of building hvac systems: A review. *HVAC&R Research*, 14(1):3–32, 2008. doi: 10.1080/10789669.2008.10390991. URL http://www.tandfonline.com/doi/abs/10.1080/10789669.2008.10390991.

[118] Congbo Li, Qinge Xiao, Ying Tang, and Li Li. A method integrating taguchi, {RSM} and {MOPSO} to {CNC} machining parameters optimization for energy saving. *Journal of Cleaner Production*, 135:263 – 275, 2016. ISSN 0959-6526. doi: http://dx.doi.org/10.1016/j.jclepro.2016.06.097. URL http://www.sciencedirect.com/science/article/pii/S0959652616307740.

[119] E.T. Lau, Q. Yang, G.A. Taylor, A.B. Forbes, P.S. Wright, and V.N. Livina. Optimisation of costs and carbon savings in relation to the economic dispatch problem as associated with power system operation. *Electric Power Systems Research*, 140:173 – 183, 2016. ISSN 0378-7796. doi: http://dx.doi.org/10.1016/j.epsr.2016.06.025. URL http://www.sciencedirect.com/science/article/pii/S0378779616302334.

[120] OConnor John J., Robertson Edmund F., and August Ferdinand Mobius. Mactutor history of mathematics archive, 1997.

[121] J.F. Dannenhoffer and R. Haimes. Design sensitivity calculations directly on cad-based geometry. Technical Report AIAA-2015-1370, 53rd AIAA Aerospace Sciences Meeting, 2015.

[122] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C, The are of scientific computing*. Cambridge University Press, 1988. ISBN 0-521-35465-X.

[123] Ira H. Abbott and Albert E. Von Doenhoff. *Theory of wing sections, including a summary of airfoil data*. Dover Publications Inc., 1949. ISBN 486-60586-8.

[124] Robert Haimes and John F. Dannenhoffer. The engineering sketch pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry. *AIAA*, 3073, 2013.

[125] John F. Dannenhoffer and Robert Haimes. Design sensitivity calculations directly on cad-based geometry. *AIAA*, 1370, 2015.

[126] John F. Dannenhoffer. Opencsm: An open-source constructive solid modeler for mdao. *AIAA*, 0701, 2013.

PENGCHENG JIA

Phone: (315) 887 0889
Email: jpengcheng@yahoo.com
Address: 238 Link Hall, Syracuse University, NY13244

SUMMARY

- Ph.D. in Mechanical Engineering with 5 years optimization, computational fluid dynamic and energy experiences
- M.Sc. in Computer Science with 3 years algorithm, artificial intelligence and software development experience
- M.Sc. in Engineering Thermo Physics with 2 years power plant operation, combustion and heat transfer analysis experience
- B.Sc. in Heating and Dynamics Engineering with 4 years HVAC, refrigeration and heat transfer experience.
- 3 years geometry modeling, CFD and optimization experience for AIR FORCE RESEARCH LABORATORY
- 2 years automotive product (IC engine) energy efficiency and calibration experience at BEIQI FOTON MOTOR CO.
- Published 7 papers in the optimization, geometry modeling, combustion and heat transfer related areas.

EDUCATION

- **Ph.D. in Mechanical and Aerospace Engineering SYRACUSE UNIVERSITY, US 2017**
  Concentration: Geometry Modeling; Reverse Engineering; CFD; Optimization
- **M.Sc. in Computer Science SYRACUSE UNIVERSITY, US In Progress**
  Concentration: Algorithm Design, Artificial Intelligence and Machine Learning
- **M.Sc. in Engineering Thermo Physics BEIJING JIAOTONG UNIVERSITY, CHINA 2012**
  Concentration: Heat Pump Design, IC Engine, Heat Transfer and Energy Conservation
- **B.Sc. in Heat and Dynamics Engineering TIANJIN UNIVERSITY OF COMMERCE, CHINA 2010**
  Concentration: HVAC, Refrigeration System, Thermodynamics and Heat Transfer

PROFESSIONAL EXPERIENCE

AIR FORCE RESEARCH LAB Syracuse University – Syracuse, NY – Aug. 13 – Dec 16
Research Assistant

- Developed the artificial intelligence algorithms for generating parametric model from a cloud of points (Geometry Modeling)
- Analyzed and improved the optimization/search algorithm that is used for large data mining (Nonlinear Regression)
- Analyzed and developed the classification algorithm for multiple-component geometry generation (Reverse Engineering)

- Developed geometry generation program for Engineering Sketch Pad (CAD/CAE software) in C/Java
- Developed the sparse techniques for solving the large linear equations.

SYRACUSE UNIVERSITY Syracuse, NY - Aug. 12 - May 16
Teaching Assistant

- Assisted teaching three Mechanical Engineering courses in Syracuse University, which were Statics, Dynamics, and Thermodynamics.
- Graded the homework for the whole class (140 students) twice per week and made solutions for students
- Provided recitations for reviewing the lecture, discussing the exercise and answering questions for all courses listed above

BEIQI FOTON MOTOR CO. Beijing Jiaotong University - Beijing, China – Oct. 10 – Aug. 12
Research Assistant & Calibration Engineer

- Analyzed the characteristics of pressure fluctuation in common rail of diesel engines by Fluent (Fuel Injection System, CFD)
- Analyzed the performances of IC engines and model based control design (MBD) for ignition strategies system (Ignition System)
- Collected and analyzed the data of engine (BENCH & OBD), via Measure Data Analyzer (MDA) in INCA (Calibration)
- Optimized the engine control algorithms (ECS) to improve the combustion efficiency in the cylinders, via MATLAB/Simulink (MBD)

DATANG INTL POWER GENERATIONAL CO. Beijing, China – Oct. 10 – Aug. 11
Research Assistant

- Analyzed the effects of flow velocity, voltage and specific resistance on droplet mass concentration distribution (CFD)
- Compared the defogging performance during applying the DC and AC voltages
- Analyzed the priority of impact for voltage, flow velocity and specific resistance on defogging efficiency

CHINA ACADEMY OF RAILWAY SCIENCES Beijing, China – Oct. 10 – Aug. 11
Research Assistant

- Developed a model of how the geothermal load affects ground temperature recovery.
- Developed a 3D coupled Finite Element Model (FEM) for a vertical geothermal heat exchanger.
- Developed a monitor and evaluation system for the geothermal heat pump applied into railway stations.

GARRISON COMMAND OF TIANJIN Tianjin, China - Oct. 09 - Jul. 10
HVAC System Designer

- Designed the structure and the piping system of HVAC, and calculated the hydraulic diameter for each pipe.
- Developed a prediction model of HAVC load for the individual building management system

(BMS)

- Simulated the heat flow and energy efficiency for the whole building by CFD (Fluent)

PUBLICATION & PAPERS

- **Generation of Parametric Aircraft Models from a Cloud of Points (via optimization methods)**
  54rd AIAA Aerospace Sciences Conference, 2015
- **Combustion Characteristics of Natural Gas in High Temperature and Oxygen-rich Condition**
  Engineering Thermo physics Conference, 2013
- **Combustion Characteristics of Gas at High-temperature and Rich-oxygen Conditions: Modeling and Experiment (design and setup the experiment control system)**
  Master Thesis in Beijing Jiaotong University, 2012
- **Probability of Electric Vehicle Promotion and Improvement of Electricity Generation Structure (via optimization methods)**
  Honorable Mention Paper in Interdisciplinary Contest in Modeling, 2011
- **Influence of the Wind Break Wall Structures upon Fluid Dynamic in Direct Air Cooled System**
  Journal of Beijing Jiaotong University, 2012
- **Characteristics of Radiation Heat Transfer of Typical Biomass Burning in Furnace**
  Journal of Engineering Thermo physics, 2011
- **Heat Transfer Characteristics of Coke Oven Gas Combustion in the Glass Melting Furnace**
  Engineering Thermo physics Conference, 2012

TECHNICAL SKILLS

- HVAC
- I.C. Engine
- Fluent & Star-CD
- Artificial Intelligence
- Computer Graphics/ OpenGL
- Algorithm & Data structure
- Operating System
- Heat Transfer
- C & C++ Programming
- MATLAB & Simulink
- Python & Java
- FEA & CFD